

Introducción a Android - Parte I

Introducción a Android

Competencias:

- Conocer el contexto social
- Conocer el contexto laboral

Introducción

Android es un sistema operativo para smartphones, tablets, smartwatches, televisores, autos y cualquier dispositivo periférico asociado al *internet of things* (internet de las cosas), que nos permite interconectar todo tipo de contenidos y programas desarrollados para esta plataforma. En analogía, Android es para estos dispositivos, lo que windows ha sido para las computadoras de escritorio y notebooks durante mucho tiempo.

Aprender a desarrollar aplicaciones, te permite tener más y mejores oportunidades de trabajo, ya que cada día se hace más imprescindible para las personas, proyectar y establecer sus negocios a través de este medio digital.

Como desarrollador android tendrás la opción de trabajar para una empresa, ofrecer consultorías, crear tu propio negocio basado en la tecnología, dar soluciones a problemáticas cotidianas a través de aplicaciones, enseñar a otros, pertenecer a grupos, y además, tener la opción de trabajar en cualquier país del mundo, dado que las vacantes para este tipo de cargos son internacionales.

Definiciones y usos

Android ofrece un completo framework (herramientas y librerías de programación) que te permite crear apps y juegos innovadores para dispositivos móviles en los lenguajes de desarrollo Java y Kotlin.

Las aplicaciones en Android se compilan como una combinación de diferentes componentes que se **pueden invocar de manera individual**. Por ejemplo, una actividad individual proporciona una sola pantalla para una interfaz de usuario y un servicio realiza tareas de manera independiente en segundo plano.

Si recién te inicias en el desarrollo de aplicaciones en Android, es importante que comprendas los siguientes conceptos fundamentales acerca del framework de Android:

- Las aplicaciones en Android se compilan como una combinación de diferentes componentes que se pueden invocar de manera individual. Por ejemplo, una actividad individual proporciona una sola pantalla para una interfaz de usuario y un *servicio* realiza tareas de manera independiente en segundo plano.

Desde un componente puedes iniciar otro componente con una *intent*. Incluso puedes iniciar un componente en otra app, como una actividad en una app de mapas para mostrar una dirección. Este modelo proporciona varios puntos de entrada para una app y permite que cualquier app se comporte como “predeterminada” de un usuario para una acción que otras apps pueden invocar.

- Las aplicaciones se adaptan a diferentes dispositivos que utilizan Android proporciona un framework de apps adaptable que te permite ofrecer recursos exclusivos para diferentes configuraciones de dispositivos. Por ejemplo, puedes crear diferentes archivos de diseño XML para diferentes tamaños de pantalla y el sistema determina qué diseño aplicar en función del tamaño de pantalla del dispositivo actual.

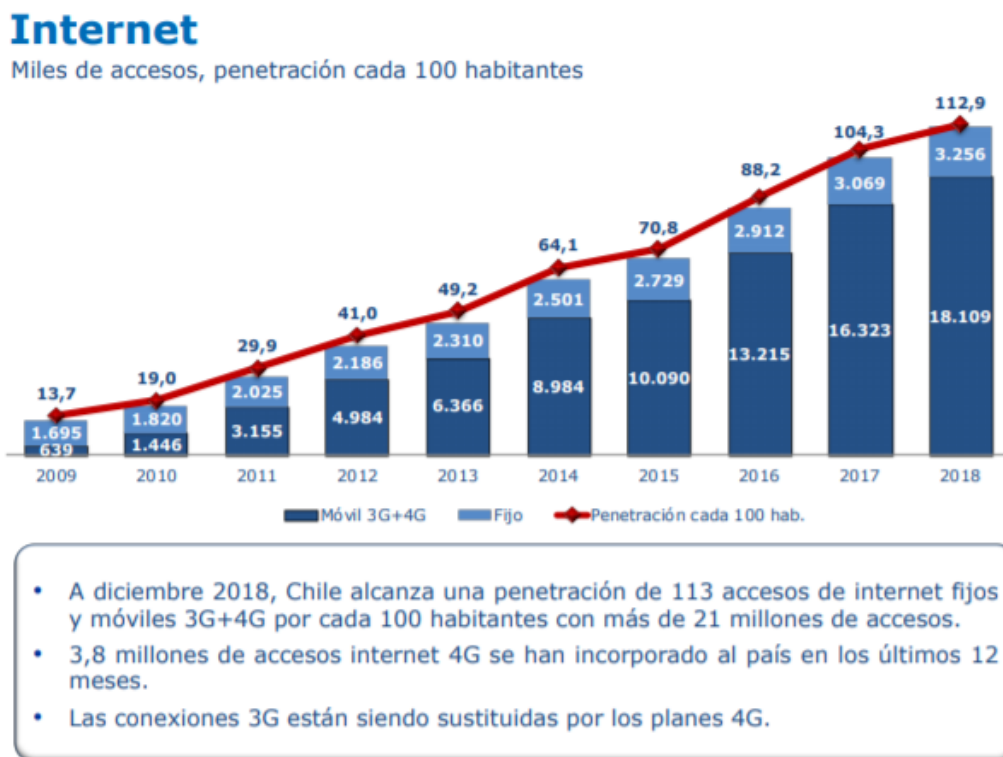
Puedes consultar la disponibilidad de funciones del dispositivo en tiempo de ejecución para averiguar si alguna función de la aplicación requiere hardware específico, como una cámara. Si fuera necesario, también puedes declarar funciones que requiera tu app para que los mercados de aplicaciones como Google Play Store no permitan su instalación en dispositivos que no admiten esas funciones.

Importancia de Android

Aprender android es una decisión muy acertada para el futuro y brinda un número de oportunidades de trabajo que no para de crecer, ofrece desarrollo profesional a largo plazo, otorga herramientas para desarrollar emprendimientos y facilita el acceso a brindar servicios para toda una sociedad.

En la actualidad el mercado global de telefonía celular es gigante, la cantidad de teléfonos celulares vendidos hasta la fecha ya sobrepasa la cantidad de personas que habitan en todo el planeta, nuestro mundo avanza hacia la personalización de servicios y tareas, al acercamiento de todas las acciones cotidianas.

Los teléfonos celulares son el canal digital más importante de la actualidad, resultando en una masificación de aplicaciones y procesos con el fin de llegar al usuario en todo momento sin importar el lugar donde esté. En la imagen 1, podemos observar la masificación del servicio de telefonía móvil en Chile, considerándose como principal medio de accesos a internet, siendo entonces el canal digital objetivo de gobiernos, empresas, instituciones, etc.



Subtel – División Política Regulatoria y Estudios



Imagen 1. Acceso a internet.

Fuente: Subtel

Android en empresas

En las empresas e instituciones, día a día se requieren más programadores Android, todo se perfila hacia la digitalización, es una tendencia marcada, razón por la cual todo están pensando en tener una aplicación propia o mejor aún muchas aplicaciones.

Principales portales de trabajo para programadores mobile:

- GetOnBoard
- Computrabajo
- Laborum
- Trabajando

Android en el emprendimiento

Las empresas nacientes denominadas “startups” que tienen conocimiento tecnológico, tienen una importante ventaja sobre aquellas que no. Si tienes en mente crear tu propia empresa y sabes construir aplicaciones móviles, tienes posibilidades de masificar tus ideas, cubrir áreas con gran demanda de tecnología o sencillamente hacer un nuevo nicho en los negocios, implementando tu plan de negocio.

Android como Freelancer

El mercado laboral ofrece muchas posibilidades de trabajo por tu cuenta. Todas las empresas requieren de tecnología en la actualidad, para llegar a sus clientes finales, sin importar su rubro o finalidad; es por esto, que si sabes programar en android, estarás habilitado para recibir cualquier requerimiento de los distintos actores de la sociedad, tanto personas naturales como jurídicas.

Por otra parte, existen muchas páginas web que se dedican a ofrecer trabajo para freelancers donde podrás solicitar tu incorporación (previo exámenes y entrevistas) con lo que también tendrás acceso a trabajo por tu cuenta a través de un mercado que controla la distribución de los requerimientos. Por ejemplo, [FreeLancer.cl](https://www.freelancer.cl) o [Workana.com](https://www.workana.com).

Tendencias globales

Android en el mercado global de la tecnología móvil lidera todas las estadísticas y se posiciona como el gigante de la industria, como nos muestra la imagen 2, el total de smartphones vendidos con sistema operativo android hasta el año 2017, alcanza la impactante cifra de 84% del total del mercado global, siendo su mayor competidor Apple con su sistema operativo IOS.

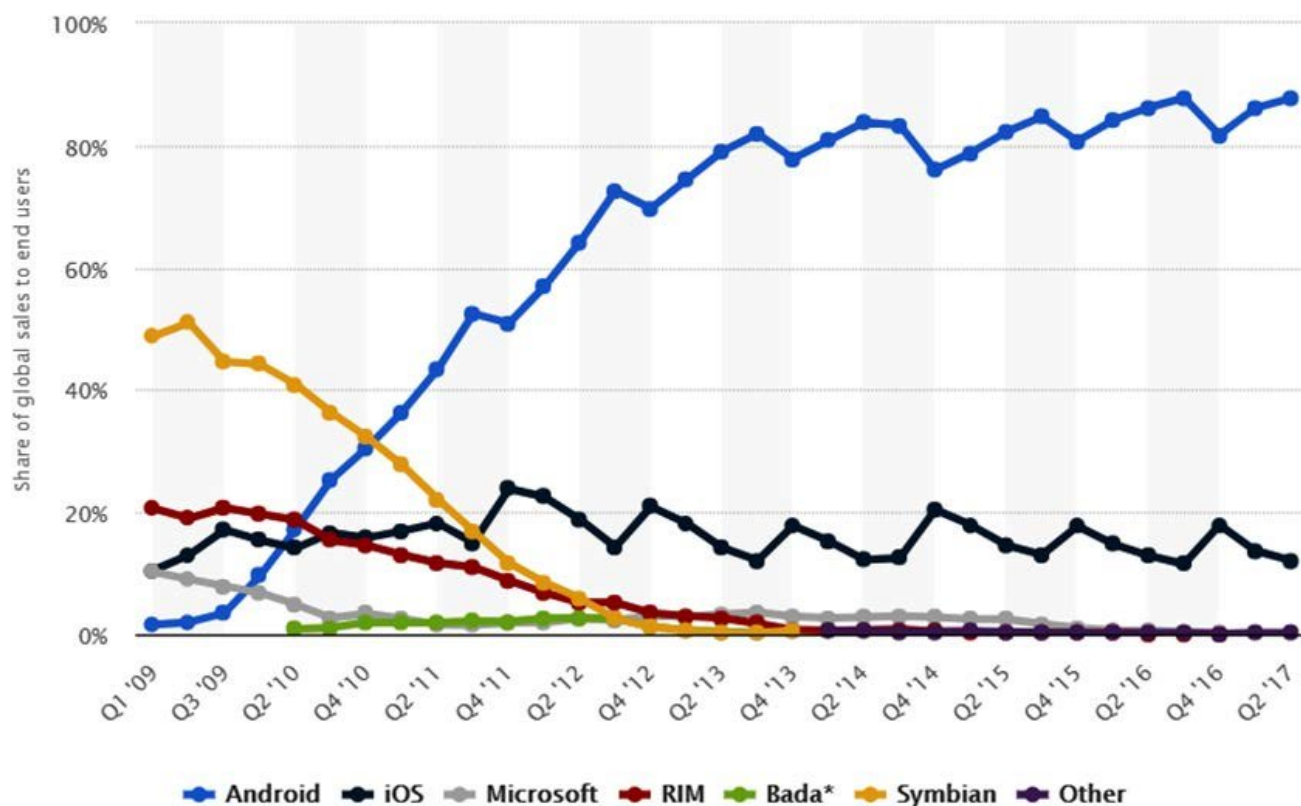


Imagen 2. Tendencias Globales.

Fuente: Statista.com

Google Developers Group







Trabajar con android también te brinda la oportunidad de integrarte a comunidades de desarrolladores que ofrecen soluciones, responden dudas, comparten información y se apoyan mutuamente.

En esta instancia se desarrollan eventos frecuentemente, generalmente participan especialistas nacionales o internacionales en distintas áreas.

Para unirse a esta comunidad independiente de [GDG Santiago](#), solo necesitas tener la voluntad de aprender.

Aplicaciones populares

Existen millones de aplicaciones desarrolladas con la tecnología android a las cuales puedes acceder a través de la Google play store y descargar alguna de ellas. Entre las más usadas se encuentran principalmente las aplicaciones de propiedad de Google (dueño android) como google maps, google drive; sin embargo encontrarás muchísimas otras más destacadas como por ejemplo las siguientes:

Logo	Aplicación
	WhatsApp - Chat, llamadas, videollamadas.
	Google Maps - Geolocalización, mapas y ubicaciones en general.
	Twitter - Mensajería, Followers, Noticias.
	Waze - Geolocalización, mapas y ubicaciones en general.
	Skype - Videollamadas, reuniones online, chat.
	Spotify - música

Entorno Integrado de desarrollo (IDE)

Competencias:

- Conocer la interfaz para la creación de proyectos en Android Studio
- Crear emuladores de teléfono en nuestra PC desde nuestro IDE.
- Asociar un sistema de control de versiones GIT a un proyecto.
- Crear aplicaciones base automáticas con el IDE y ejecutarlas en nuestro emulador.

Introducción a Android Studio

Para construir aplicaciones requerimos de herramientas y funcionalidades asociadas al sistema operativo android, necesitamos un IDE , en donde podamos escribir nuestro código de una manera ordenadas y simple, poder compilar una aplicación para que el IDE la analice y nos informe de la existencia de errores o no, hacer seguimiento a secciones de código, actualizar versiones del sistema operativo y ejecutar las aplicaciones para observar sus distintas etapas, entre otras funciones.

AndroidStudio es el IDE oficial para desarrollar y crear aplicaciones Android. Android Studio está constantemente actualizado, es moderno y lo soporta una comunidad muy grande de usuarios a nivel mundial, la empresa Google y la empresa JetBrains, que para los que no la conocen, es la responsable de los mejores IDE del mercado para distintos lenguajes de programación.

Instalación del entorno de trabajo

[Instalación guiada](#) para windows/mac/linux.



Imagen 3. Instalación de Android

Creación de un proyecto

Una vez instalado nuestro IDE Android Studio, podemos crear nuestro primer proyecto para preparar nuestro entorno de trabajo, como lo muestra el siguiente flujo de imágenes:

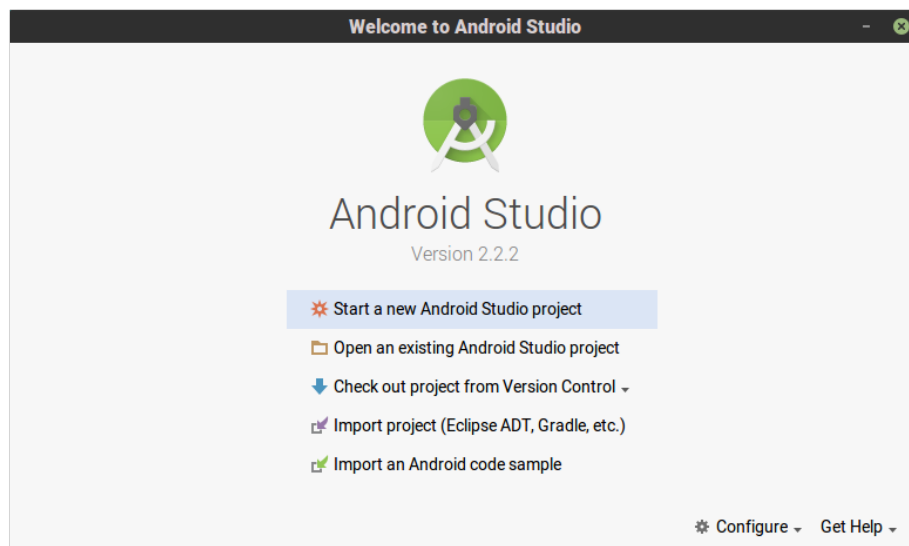


Imagen 4. Crear un nuevo proyecto Android Studio

En esta pantalla escogemos la primera opción de comenzar un nuevo proyecto Android.

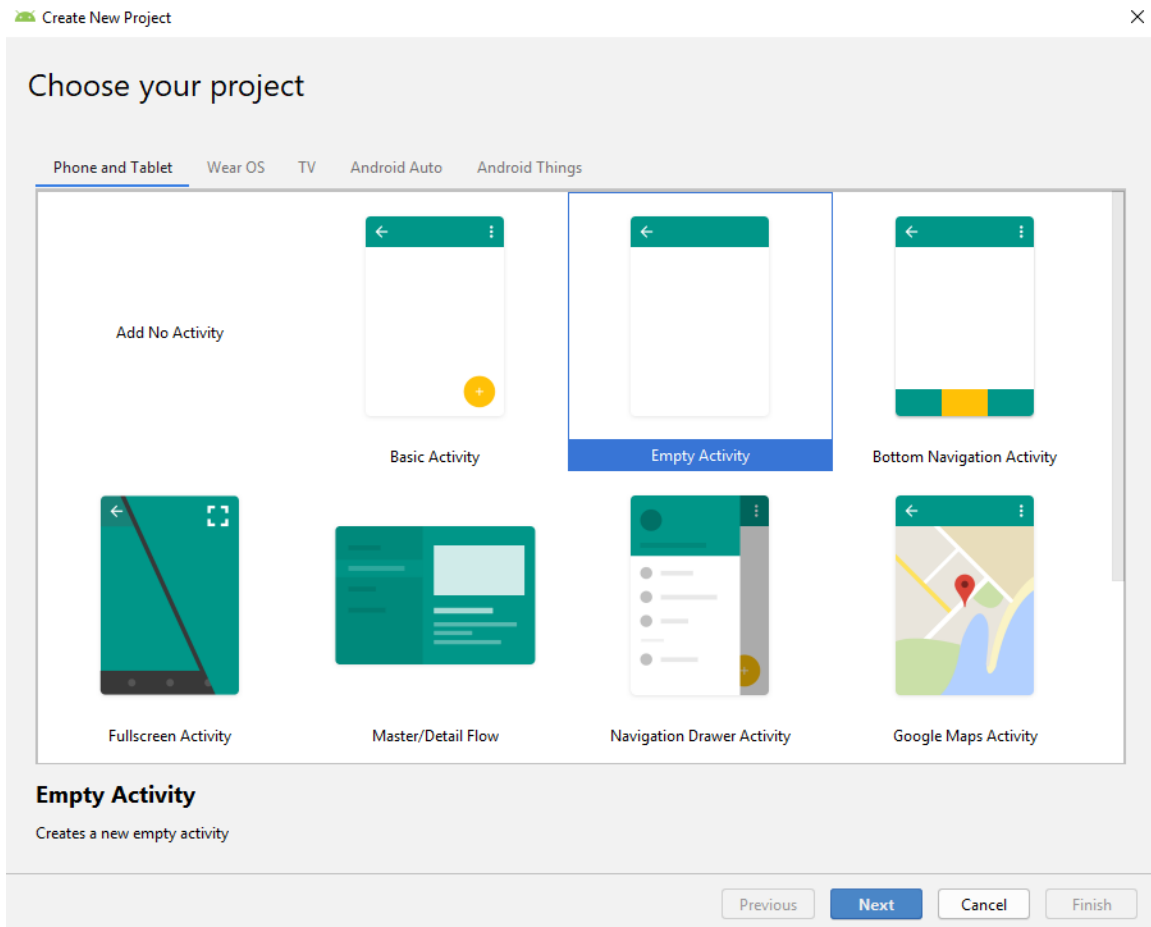


Imagen 5. Escoger una plantilla

En este punto, el IDE nos muestra una serie de plantillas iniciales para comenzar proyectos con un esqueleto de archivos ya armado en distintas modalidades, en esta oportunidad seleccionaremos **Empty Activity** y hacemos click en siguiente.

Create New Project

Configure your project

Name
MiPrimerProyecto

Package name
cl.desafiolatam.android

Save location
C:\workspace_mobile\MiPrimerProyecto

Language
Java

Minimum API level
API 22: Android 5.1 (Lollipop)

Empty Activity
Creates a new empty activity

Additional options:
☐ This project will support instant apps
☐ Use AndroidX artifacts

Footer: Previous Next Cancel Finish

Imagen 6. Configurar el proyecto

En la pantalla de configuración del proyecto, ingresamos los siguientes parámetros:

- **Nombre del proyecto.**
- **Package Name:** este en particular resulta ser muy importante porque es la llave que identifica tu aplicación en las tiendas, es decir en Google Play y la App Store de apple. Por otra parte, el package name contiene keywords muy importantes para la explotación comercial de tu aplicación, por lo tanto, siempre se debe estudiar su selección para la aplicación de las mejores técnicas ASO (Application Store Optimization).

El package name de una aplicación publicada en las tiendas, ya no puede ser modificado, dado que representa el dominio de nuestro negocio, el documento de identidad de la aplicación en las tiendas.

- **Directorio** donde se guardará el proyecto.
- **Lenguaje JAVA.**

- **API** mínimo para que opere esta aplicación, es decir, que estará disponible para todos los dispositivos móviles que funcionen con el sistema operativo android 22 (Lollipop, 5.1) en adelante.

Nótese que el IDE nos deja una información importante con la selección del API 22, nos informa que de esta forma estará disponible para el 80,2% de teléfonos Android activos a nivel mundial.

De lo anterior, considerar que las actualizaciones de los sistemas operativos móviles es constante, y también el reemplazo de teléfonos por parte de los usuarios, por lo tanto, es una buena práctica desarrollar considerando siempre que el mercado evoluciona rápidamente.

- Seleccionamos **Finalizar**.

Navegación y vistas de entorno sobre el proyecto

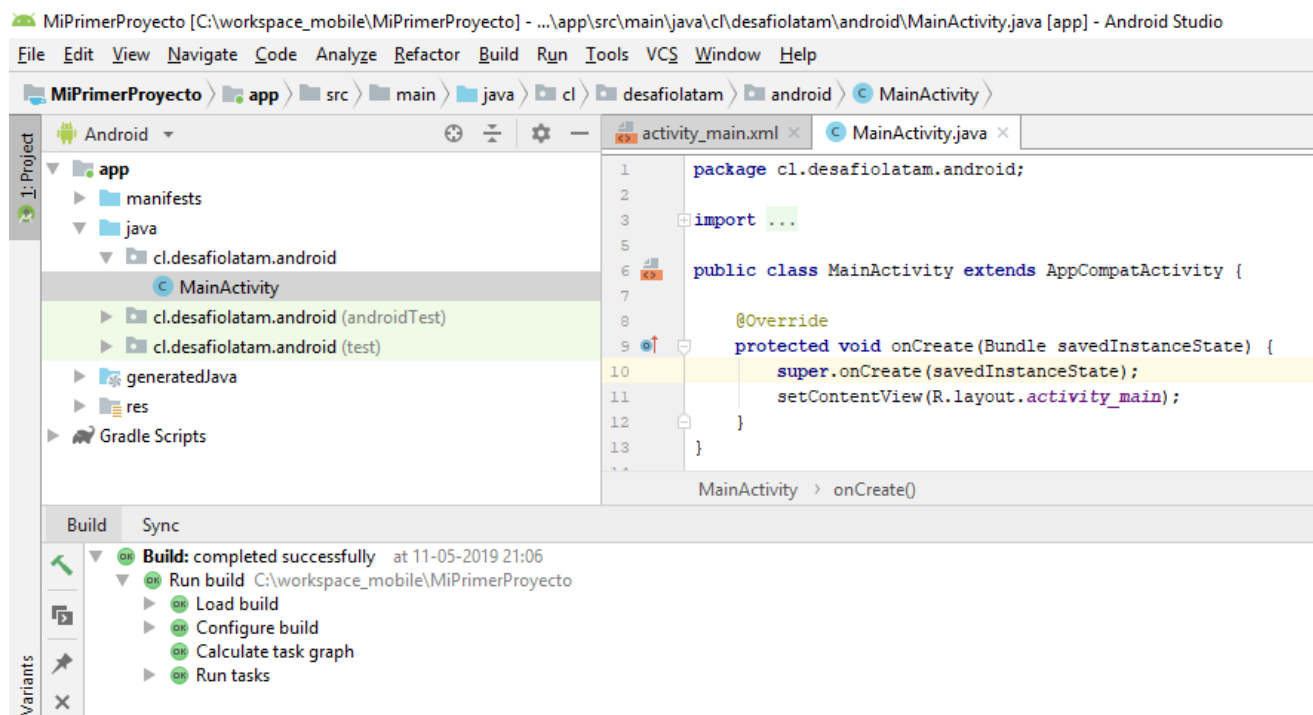


Imagen 7. Entorno del proyecto.

El IDE creará la base de nuestro proyecto, mostrando nuestra Actividad principal y una estructura de carpetas que detallaremos más adelante.

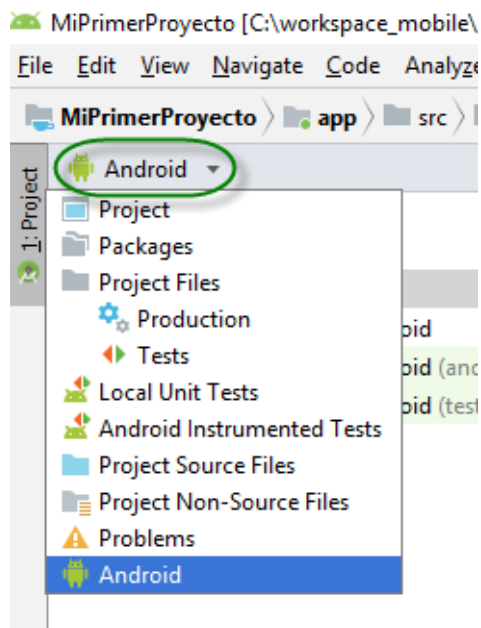


Imagen 8. Opciones de navegación.

Al hacer click sobre el combobox o spinner como lo muestra la imagen 8, se desplegarán todas las opciones disponibles que tenemos de navegación sobre nuestra estructura de carpetas dependiendo de la necesidad que tengamos en determinada fase de un proyecto.

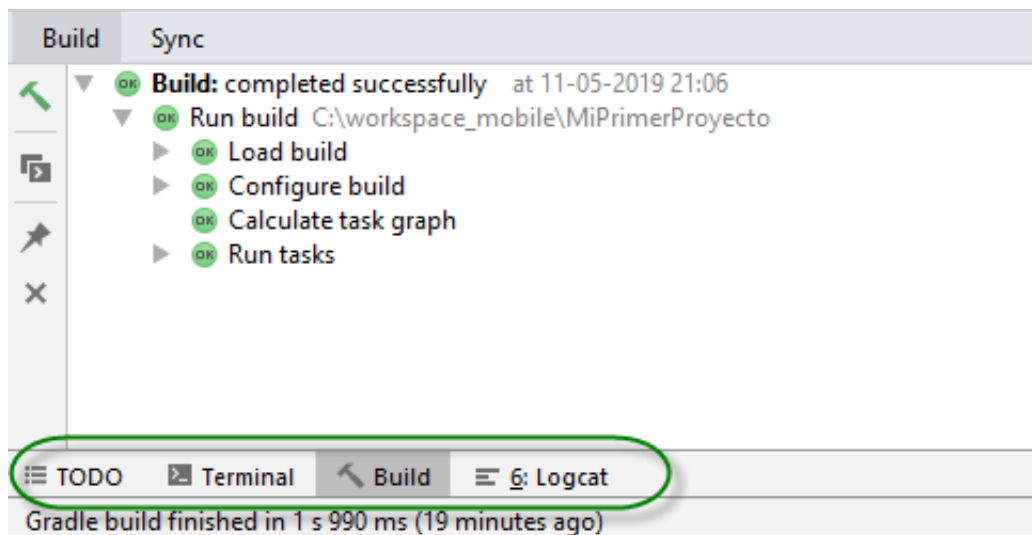


Imagen 9. Ventana inferior.

En la parte inferior de la ventana podemos identificar una subdivisión que separa los mensajes, logs y consola/terminal de la vista de archivos y estructura de proyecto, como lo muestra la imagen 9, en dónde “Terminal” es una ventana directa a la consola para tipear comandos, “Build” donde se informa cómo se completaron una compilación del programa y “Logcat” donde se muestran mensajes de log asociados al dispositivo físico o emulador que corre el programa.

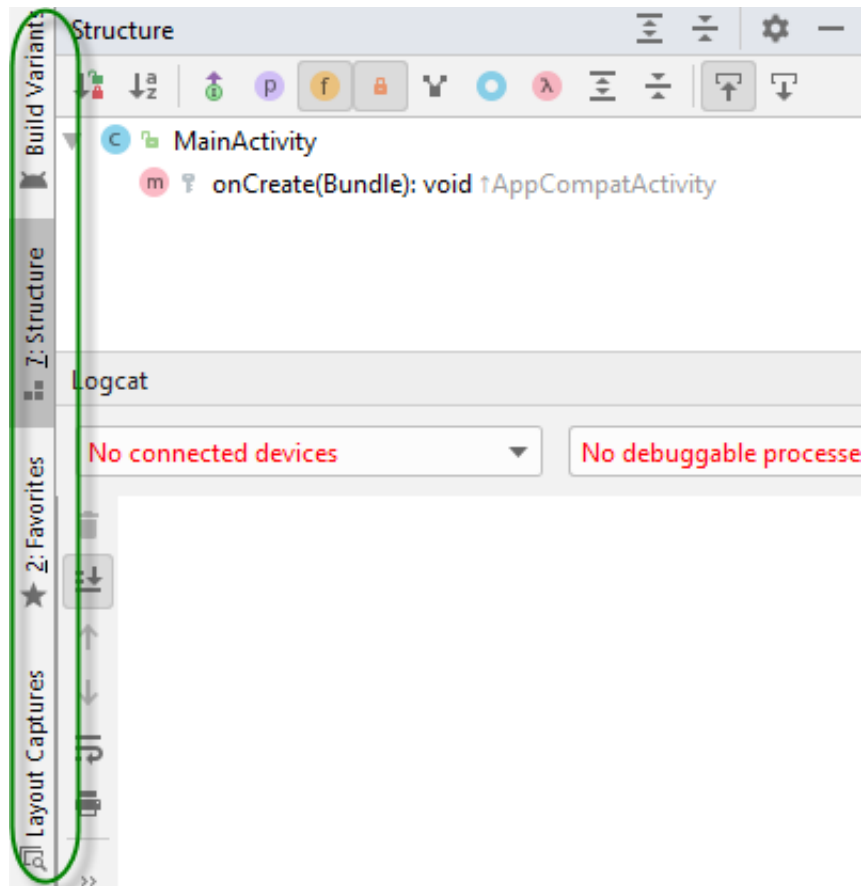


Imagen 10. Panel inferior izquierdo.

En el IDE, se observa un menú en la parte inferior izquierda el cual ofrece vistas de navegación más detalladas informando la configuración de builds disponibles, ejemplo: desarrollo, test, producción; la estructura de una clase actividad, interface, etc, y sus métodos, favoritos y layouts de vista. Ver imagen 10.

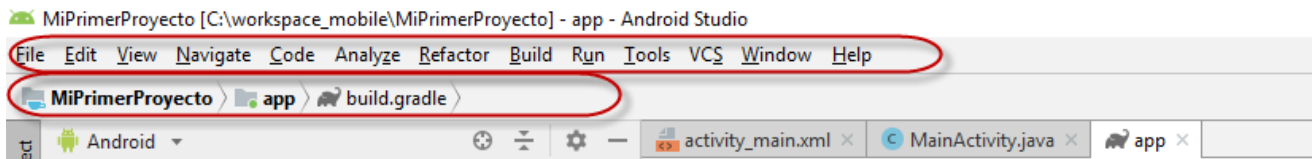


Imagen 11. Menu principal.

En la parte superior izquierda de la pantalla en el IDE, vemos un menú principal con múltiples opciones y acciones para el desarrollo de nuestras aplicaciones, entre las cuales destacamos secuencialmente las siguientes: Make build (compilar app), Run (Ejecutar app), Debug (Seguimiento de código del app) y Avd Manager (gestión de avd).

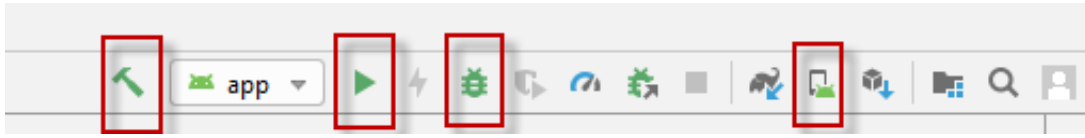


Imagen 12. Accesos rápidos

Otra sección no menos importante son los accesos directos en la parte superior derecha de nuestra pantalla en el IDE, como lo muestra la imagen 12.

Por último, recordamos que en la ventana inferior del IDE, es donde podremos visualizar el estado de compilación y ejecución de nuestra aplicación, leyendo distintos mensajes que nos proporciona el IDE.

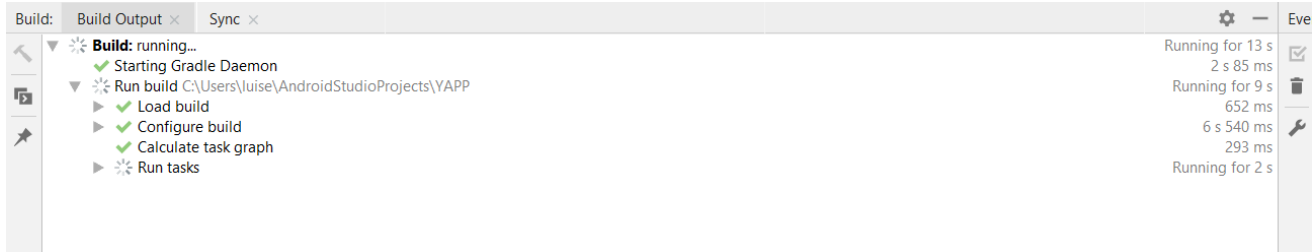


Imagen 12 B.

Software Development Kit (SDK) y SDK Manager

El SDK de android se podría decir que es el kit ya descargado e instalado con android studio, que nos permite además descargar herramientas adicionales, plugins, entre otros y que está detrás de todo el funcionamiento de nuestro IDE AndroidStudio.

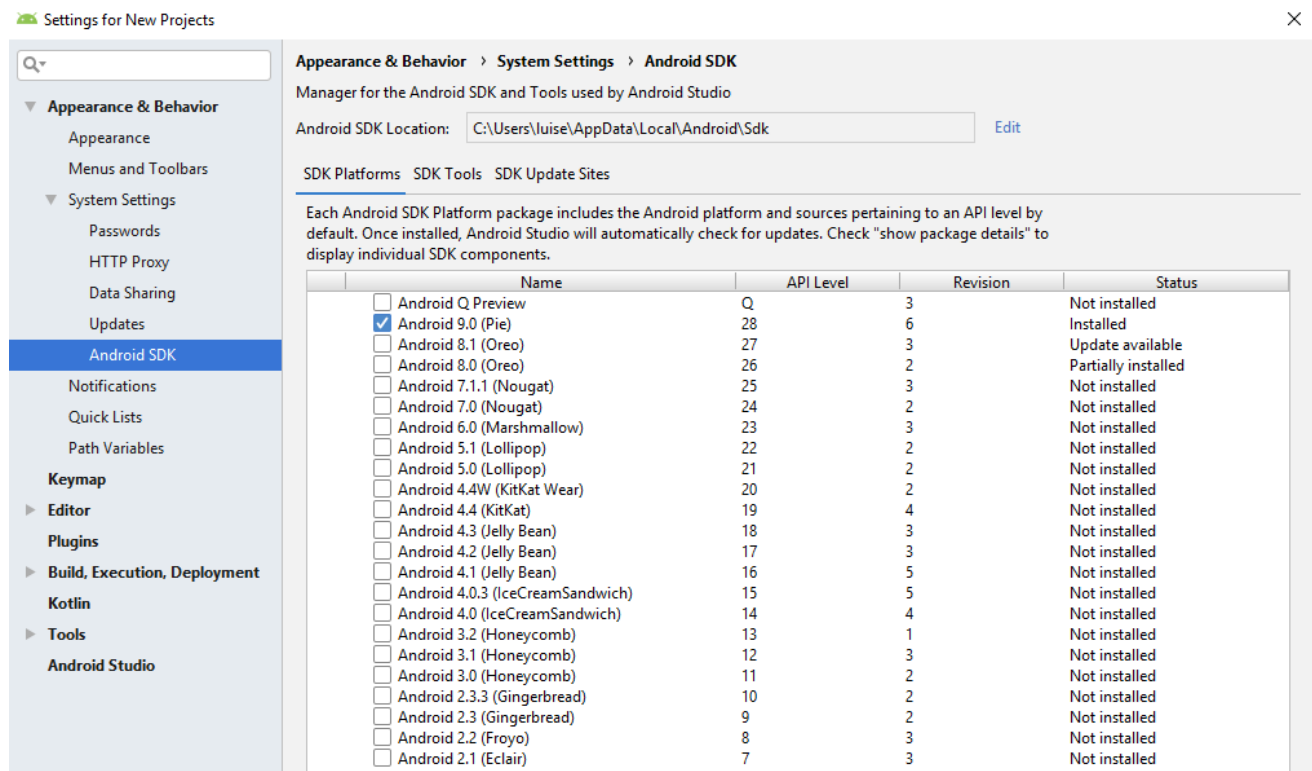


Imagen 13. Android SDK.

El SDK Manager es básicamente nuestro gestor de versiones operativas, permite que descarguemos todas las APIs de Android para trabajar con ellas en nuestro IDE, además de permitirnos configurar y descargar herramientas para optimizar dichas versiones. Para acceder al manager ir al menú superior izquierdo, seleccionando *Tools -> SDK Manager*. Ver imagen 13.

Android Virtual Device (AVD) y AVD Manager

Un AVD es un emulador de dispositivos Android (teléfonos, relojes, auto, android thing, etc) que funciona dentro de una virtual box, es decir es un sistema operativo Android dentro de un sistema operativo windows/linux o mac. Para su correcto funcionamiento es necesario:

- Descargar/actualizar las herramientas de aceleración de procesadores, como por ejemplo Haxm para Intel, esto se hace desde nuestro SDK Manager. Ver imagen 14.

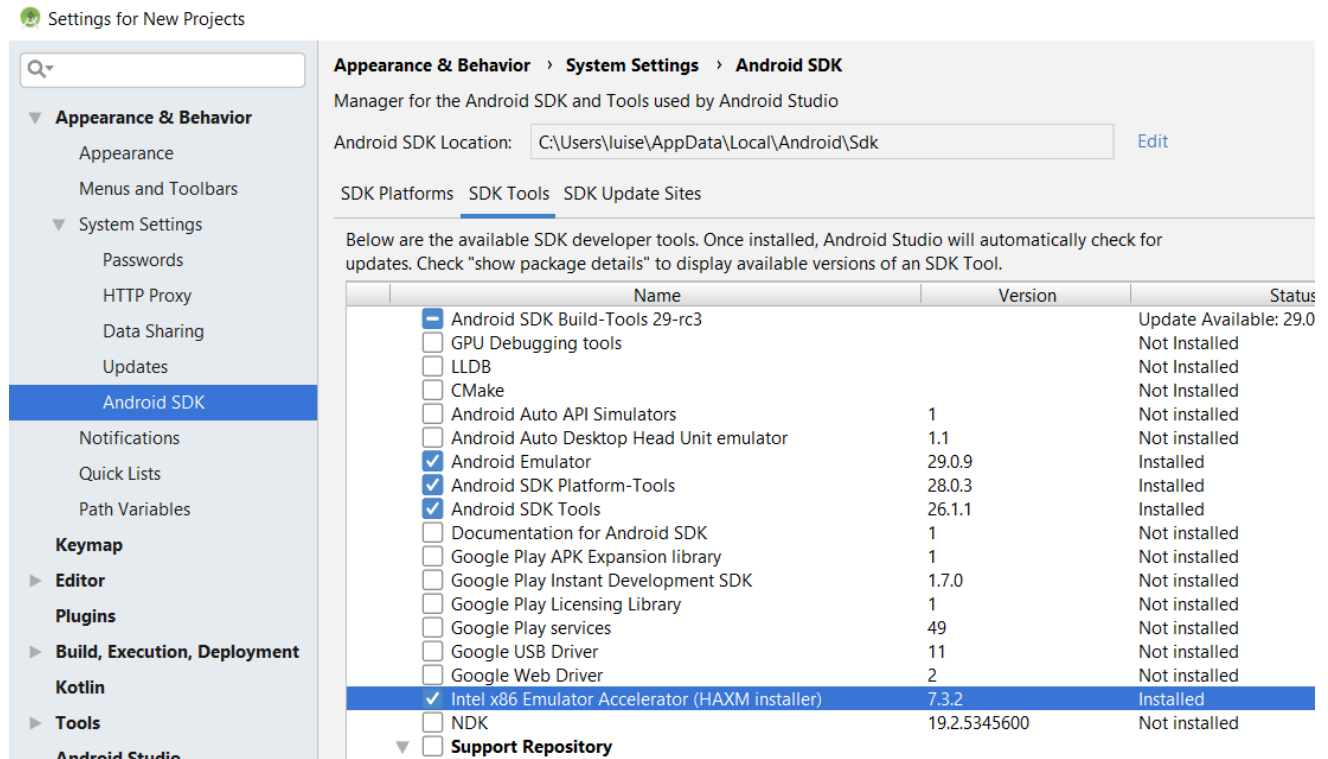


Imagen 14. SDK Manager.

- Crear un emulador desde el AVD Manager disponible en el menú superior izquierdo de nuestro IDE, opción *Tools -> AVD Manager*. Ver imagen 15.

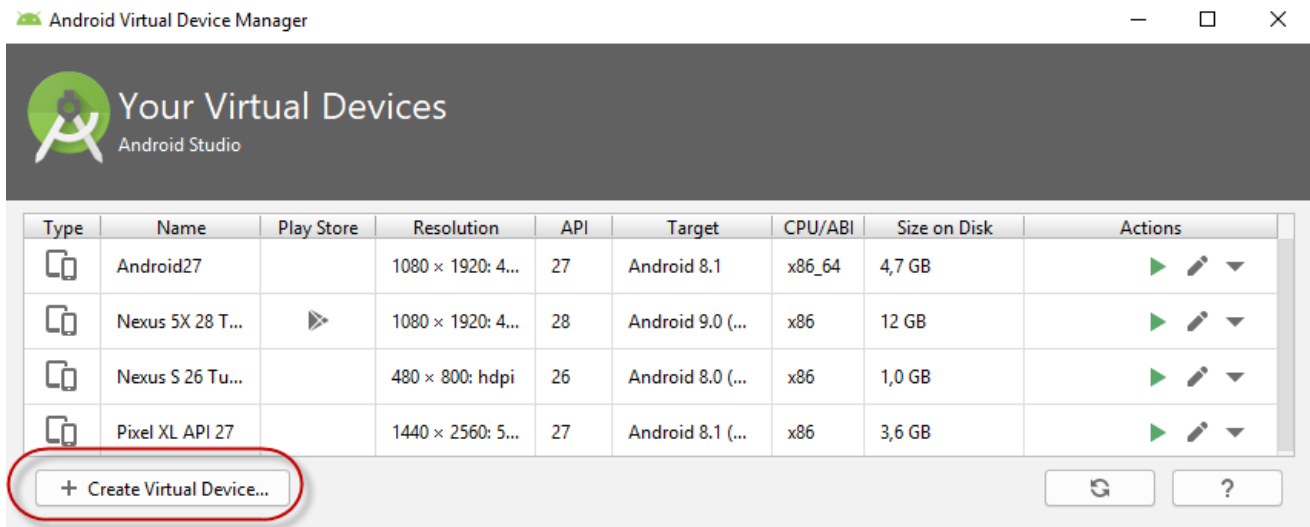


Imagen 15. Android virtual devices

- Al seleccionar “crear un nuevo virtual device” nos aparecerán más opciones de criterio como la muestra la imagen 16. En nuestro curso escogeremos Pixel 2.

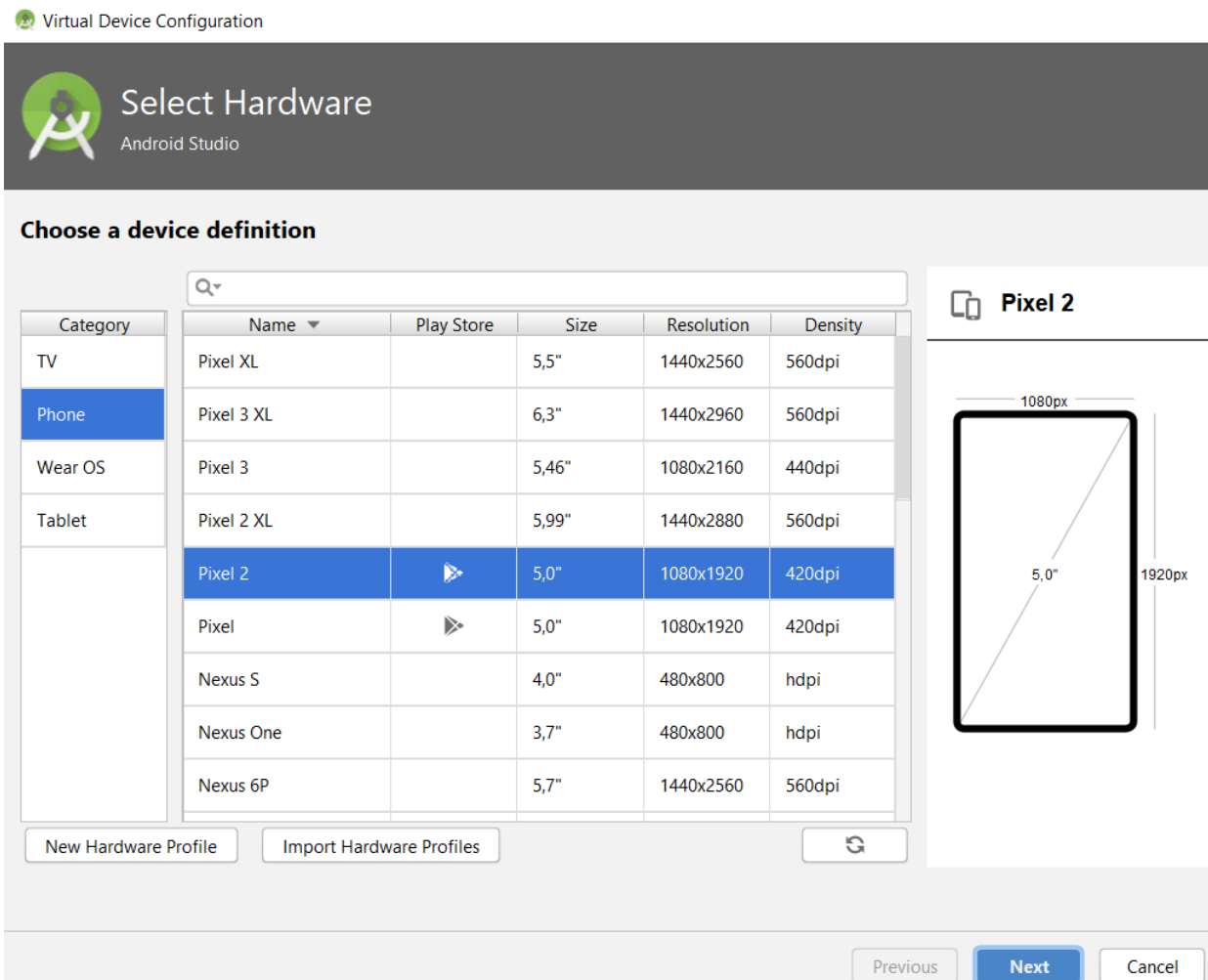


Imagen 16. Elegir el dispositivo.

- Posteriormente aparecerá una pantalla de selección del sistema operativo para el emulador del teléfono, en el cual escogeremos uno que tengamos ya disponible (si no es así seleccionaremos descargar y el IDE lo hará automáticamente), para con esto finalizar la instalación. En nuestro curso escogeremos la versión Pie 28. Ver imagen 17.

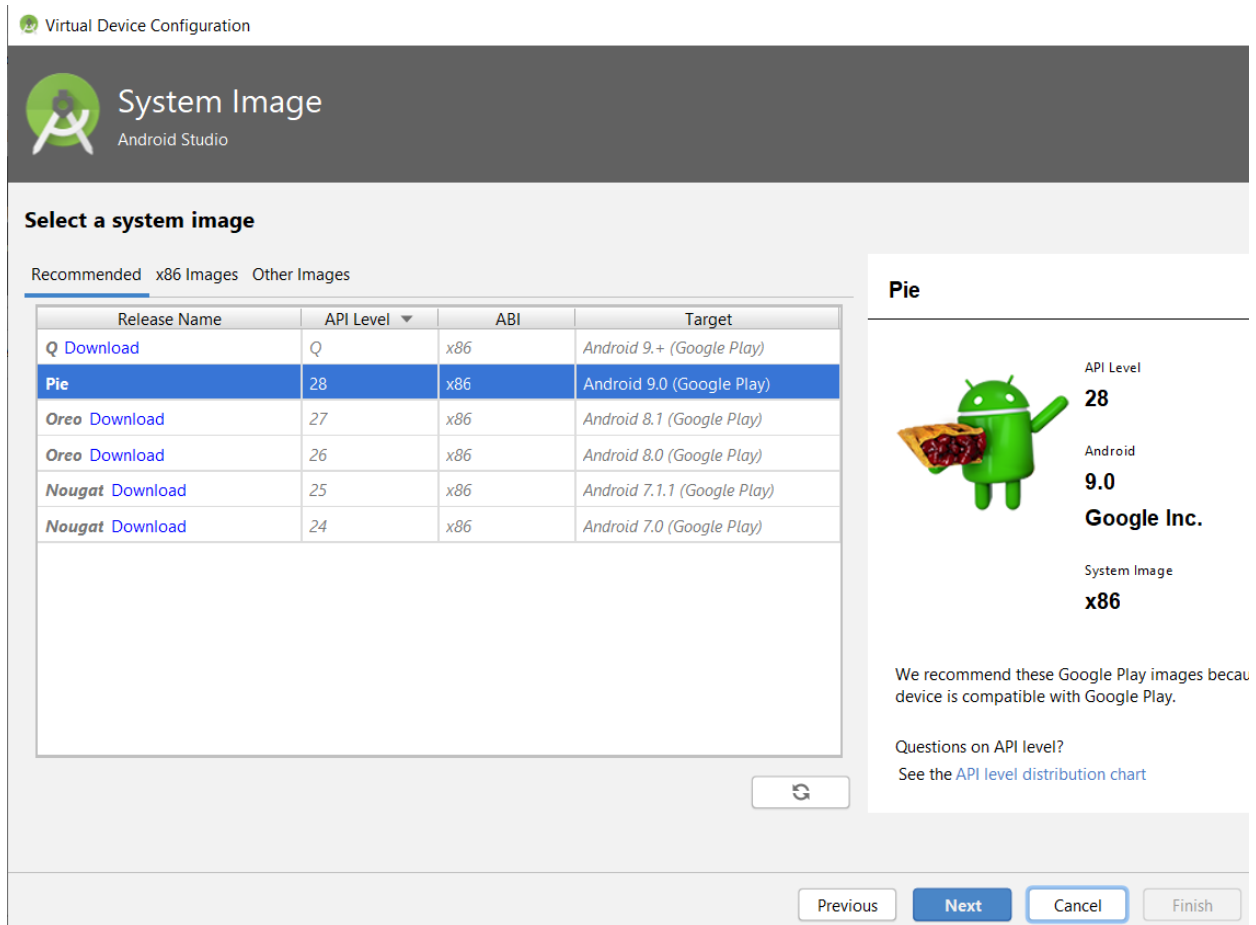


Imagen 17. Elegir la imagen del sistema operativo

Desde nuestro **Terminal** con comandos de consola, podremos ejecutar distintos comandos asociados a nuestros emuladores con el comando `adb`.

Asociando nuestro proyecto a Git**

Para asociar nuestro proyecto a GIT, existe una forma muy sencilla siguiendo estos pasos:

- Tener instalado [Git](#) en nuestra pc local.
- Desde el **Terminal** ejecutar el comando `git init`, resultando esto en la creación automática de una carpeta con nombre `.git` en la raíz de nuestro proyecto.
- Sin cerrar el IDE, luego de ejecutar el comando anterior, **podemos visualizar que se ha integrado correctamente el controlador de versiones git a nuestro proyecto** como aparece una opción como lo muestra la imagen 18.

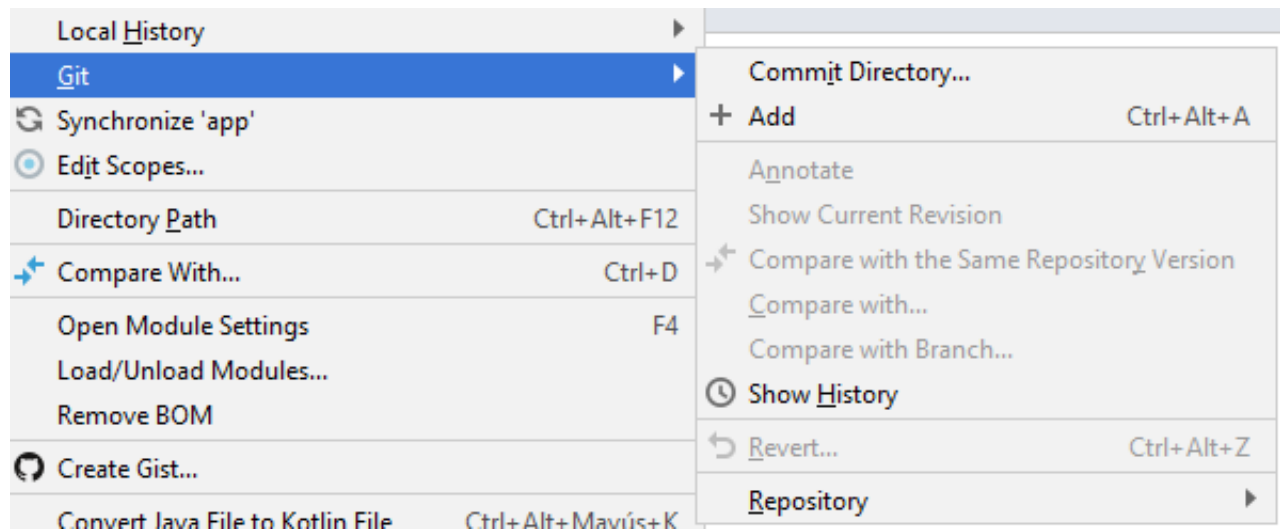


Imagen 18. Se integra Git a nuestro proyecto.

Si fuese el caso, de importar un proyecto con un `git pull`, al abrir el proyecto en el IDE, ya estaría la opción de trabajar con Git de manera integrada como hace referencia la imagen 18.

Con **git** asociado al proyecto, el IDE Android Studio siempre preguntará si se desea agregar a git un archivo nuevo que haya sido recientemente creado. Tenemos la opción de realizar acciones adicionales propias del ciclo de control git, las cuales profundizaremos en la próxima unidad.

Ejercicio 1: Ejecutar una aplicación localmente.

Para ejecutar la aplicación `MiPrimerProyecto` que nos ha creado el IDE automáticamente, teniendo ya nuestro emulador virtual creado, lo hacemos de la siguiente forma:

- Ir al menú superior izquierdo, opción Run -> Run 'app'.
- Seleccionamos el dispositivo virtual a utilizar para desplegar la aplicación.

Si está todo correctamente instalado, se debería de visualizar un teléfono con una aplicación activa con un mensaje centrado "Hello World" como lo muestra la imagen 19.

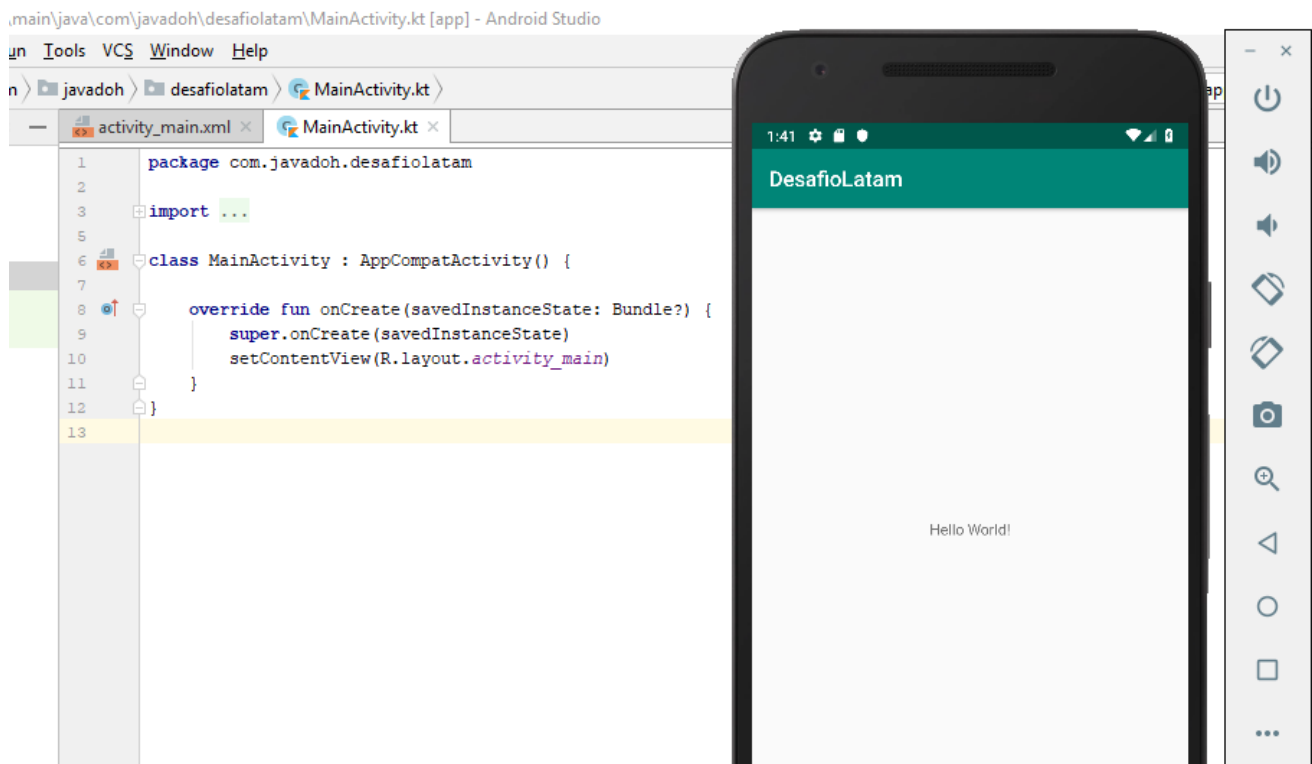


Imagen 19. Nuestra primera app

Estructura y composición de un proyecto Android

Competencias:

- Distinción de lenguajes de programación, archivos y sus funciones
- Entender la estructura de un proyecto.
- Agregar recursos de imágenes y assets a nuestro proyecto
- Configurar archivos Gradle y dependencias de un proyecto
- Conocer la clase R de indexación de recursos
- Agregar una dependencia al proyecto

Introducción

A continuación nos dedicaremos a explorar la organización y estructura de los archivos en un proyecto Android, identificando donde se ubican secciones principales como las vistas, actividades, archivos de configuración, imágenes, layouts y strings.

Estructura y archivos de vista

Dentro de la carpeta raíz de nuestro proyecto Android, podemos localizar una carpeta llamada “res” en `/app/src/main/res`, en la cual se encuentran los contenedores (layout) de nuestras vistas como lo muestra la imagen 20.

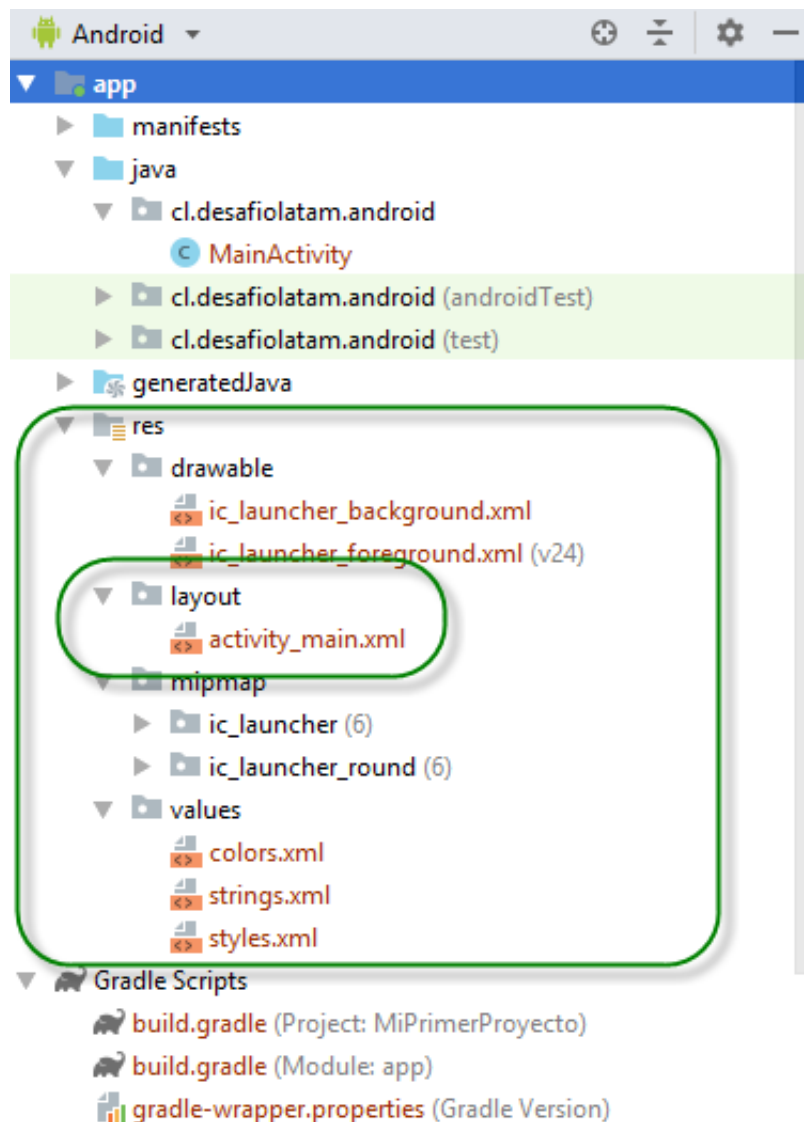


Imagen 20. Layouts.

En esta carpeta res, también se ubican los archivos de imágenes en drawable, video y sonido en la carpeta raw, iconos en mipmap y parámetros de estilo y textos estáticos dentro de la carpeta values.

Un archivo tipo layout es un archivo de formato xml, donde las vistas son agregadas con etiquetas predefinidas para su uso. Ver imagen 21.



Imagen 21. Revisando el activity_main.xml

Si prestamos atención a la imagen anterior, nos podemos dar cuenta de que el archivo activity_main.xml declara lo siguiente:

- Un contenedor de tipo ConstraintLayout con parámetros de ancho y alto cubriendo toda la pantalla (match_parent) y la actividad asociada a la vista.
- Una vista tipo TextView con parámetros de ancho y alto que solo ocupen lo que su contenido de texto requiera, un mensaje con la cadena “Hello Wordl!” que termina siendo el mensaje que visualizamos previamente (Entorno Integrado de Desarrollo IDE) al ejecutar la aplicación como hace referencia la imagen 19.

Las vistas son asociadas a las actividades desde una clase de actividad. La forma más básica de hacerlo es desde el método onCreate(), seguido del setContentView(Layout), como lo hace el siguiente código:

```

public **class** **MainActivity** **extends** **AppCompatActivity** {

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

```


Estructura y archivos de clase

Los archivos de clase que contienen código en lenguaje de programación java, se encuentran ubicados desde la carpeta raíz de nuestro proyecto hasta la ruta /app/src/main/java/ como lo muestra la imagen 22.

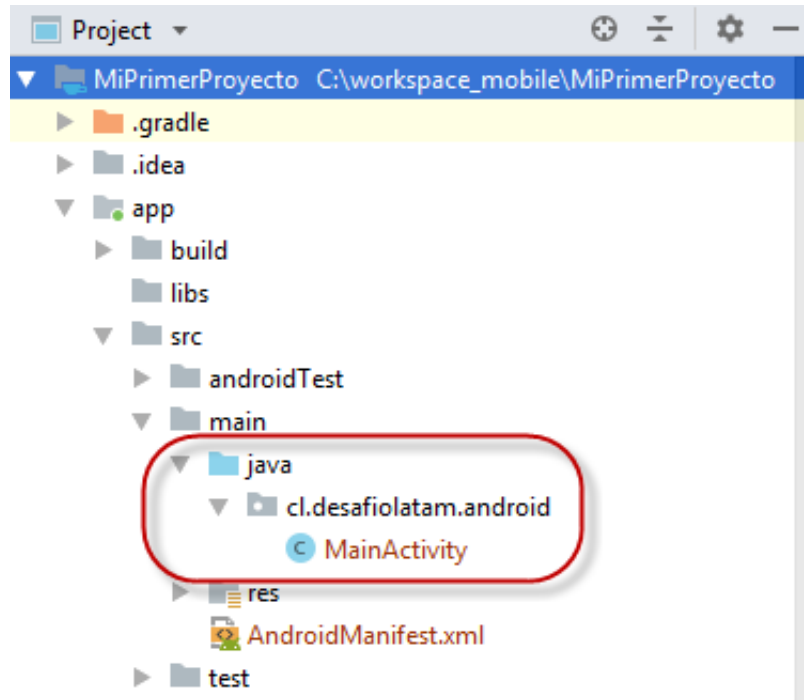


Imagen 22. Ruta hasta el MainActivity.

También encontraremos archivos java en la carpeta `test` y `androidTest`, el primero pruebas unitarias y el segundo para pruebas instrumentales o de flujo de pantallas.

Estructura y archivos de configuración

- Los archivos de configuración son múltiples y los podemos clasificar de la siguiente manera:
- Archivos de configuración del IDE ubicados en la carpeta `.idea`.
- Los archivos asociados al Gradle y todo el sistema de dependencias.
- El `proguard-rules` que cumple con agregar seguridad y permitir la ofuscación del código.
- Los archivos `.iml` que contienen parámetros asociados a nuestro proyecto.
- Todos aquellos archivos de configuración asociados a frameworks y librerías de terceros.

Archivos de recursos y assets

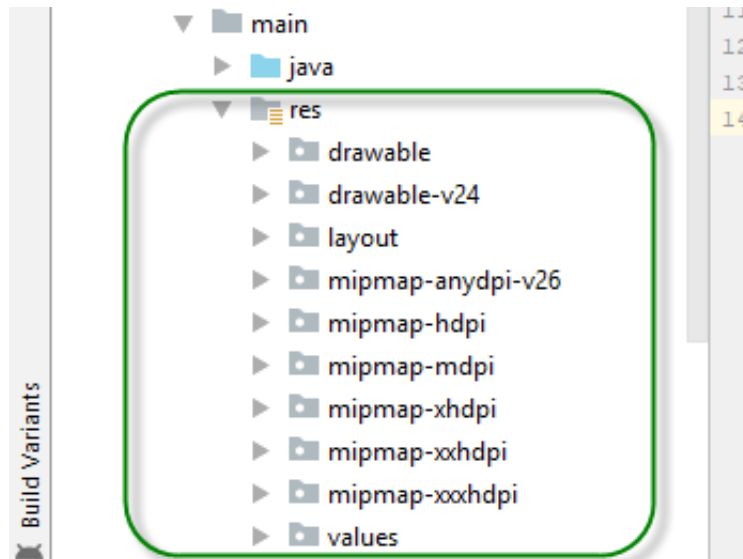


Imagen 23. Recursos.

Dentro de la carpeta `res` se pueden clasificar los siguientes tipos de archivos en subcarpetas de la siguiente manera:

- **Imágenes:** se colocan en la carpeta `Drawable` y es buena práctica generar al menos cuatro archivos con distintas densidades (`mdpi`, `hdpi`, `xhdpi`, `xxhdpi`) o importar vectores que también soportan distintos tamaños de pantallas y ocupan menos espacio, justo de la misma forma como lo hace automáticamente un `mipmap` con el icono de la aplicación.
- **Archivos personalizados de estilo, animaciones:** estos archivos suelen ser generados como un `xml` y dentro de la carpeta `drawable`, ejemplo: un `shape` base para un botón.
- **Mipmap:** Aquí solo residen iconos y preferiblemente se limita al icono principal de la aplicación, por recomendación de Google.
- **Values:** esta carpeta es muy importante, aquí se encuentran los archivos, `strings.xml`, `colors.xml`, `styles.xml` donde se declaran cadenas de texto, colores y estilos respectivamente, con el fin de ser reutilizados en distintas secciones de nuestra app.
- **Layouts:** es la carpeta contenedora de las vistas de la aplicación, como ya lo hemos descrito anteriormente.

Los `assets` en Android Studio son archivos de cualquier tipo de formato, es decir, la carpeta `assets` te permite cualquier tipo de archivo, por lo general, aquellos que no puedan ser ubicados en la carpeta `res`, un ejemplo serían archivos especiales de gráficas para juegos. La carga de estos recursos es más lenta que aquellos que se ubican en la carpeta `res`.

Gradle y dependencias

Gradle es un paquete de herramientas de compilación avanzadas, para automatizar y administrar el proceso de compilación, y al mismo tiempo definir configuraciones de compilación personalizadas y flexibles. Android compila recursos y código fuente de la app y los empaqueta en una APK (Android Application Package, en sus siglas en inglés) que puedes probar, implementar, firmar y distribuir.

En nuestro proyecto, podemos identificar dos archivos build.gradle que son fundamentales en nuestra configuración.

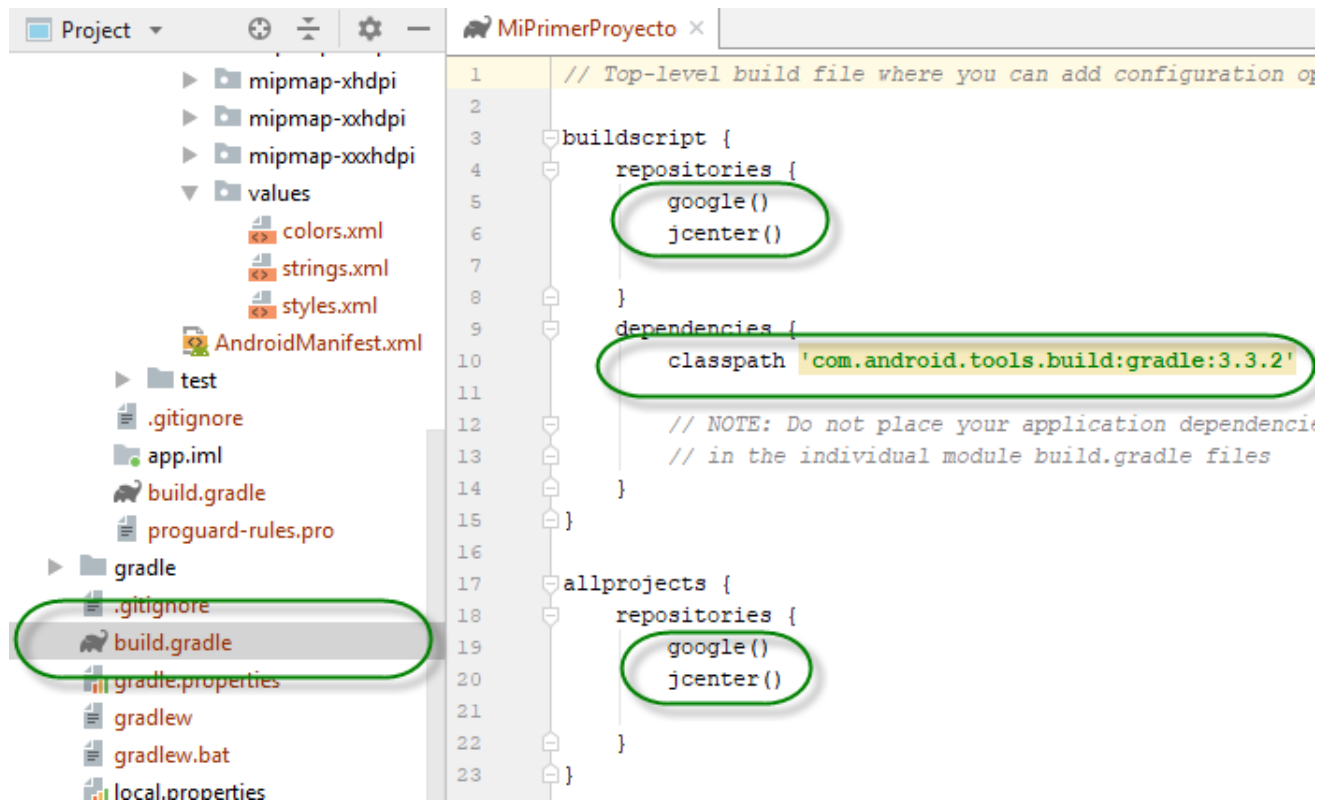


Imagen 24.Gradle.

El `build.gradle` que reside en la carpeta raíz de nuestro proyecto sirve para parametrizar qué tipo de repositorios remotos usaremos para obtener librerías y objetos disponibles para la comunidad. Como se muestra en la imagen 24, podemos usar `google`, `jcenter` o también agregar `maven`, entre otros. El `classpath com.android.tools.build:gradle:3.3.2` destacado en la imagen, se usa para indicar que versión de gradle ocuparemos para compilar nuestra app.

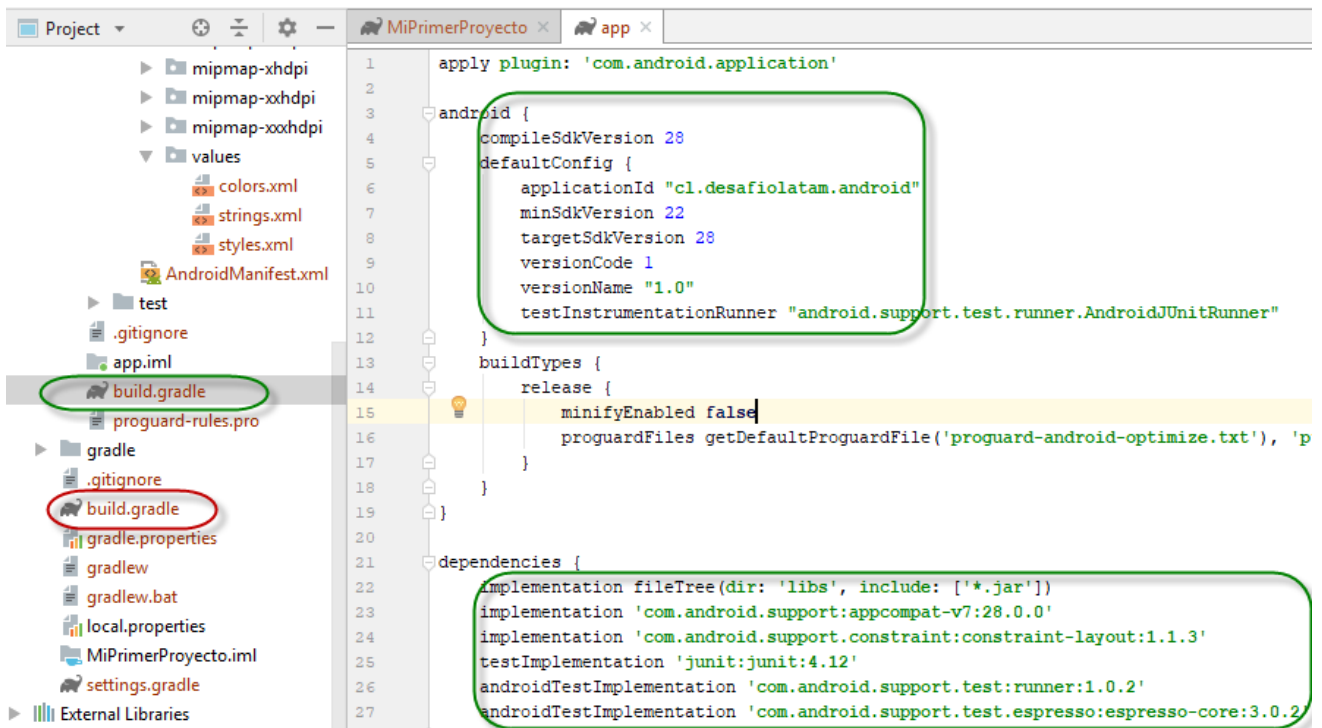


Imagen 25. build.gradle.

Existe otro archivo build.gradle, esta vez ubicado dentro de la carpeta /app, en el cual se ingresa información muy relevante de nuestra aplicación descrita y clasificada de la siguiente manera:

- **compileSdkVersion:** se ingresa el número de versión del sistema operativo android que se usará para empaquetar nuestra aplicación en un archivo .apk.
- **minSdkVersion:** Indica la versión mínima del sistema operativo Android que debe tener un dispositivo para instalar nuestra aplicación. En el caso de que un dispositivo tenga una versión menor a la declarada, la google play store nos mostrará un mensaje que indicará no podemos instalar la aplicación debido a una versión antigua del sistema operativo.
- **targetSdkVersion:** es importante la versión que parametrizamos en este parámetro para desarrollar código y utilizar funcionalidades nativas, es nuestra api base con la cual construimos la aplicación. Android, en la actualidad exige que la versión utilizada en este parámetro sea a partir de la api 26.
- **versionCode:** Cada vez que subimos una actualización a la Google Play, esta debe ir con un número mayor de versión al ya existente.
- **versionName:** solo descriptivo de uso comercial (es el número de versión que verá el usuario en Google Play).
- **dependencias:** es aquí donde ingresamos librerías y frameworks para ser usados en nuestra aplicación.

A continuación haremos un primer ejemplo práctico importando a nuestro código la librería de estilos material design para ser usada posteriormente.

Ejercicio 2: Importar la librería [Material Design](#) en nuestro archivo Gradle.

1. Abrimos nuestro archivo `build.gradle` ubicado en la carpeta `/app` y agregamos el siguiente código dentro de nuestras dependencias:

```
dependencies {  
    // ...  
    implementation 'com.google.android.material:material:1.1.0-alpha06'  
    // ...  
}
```

2. Sincronizamos nuestro gradle con esta acción haciendo click en **Sync Now**, como lo muestra la imagen 26.

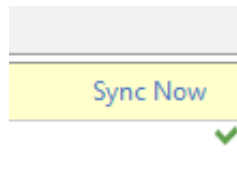


Imagen 26. Sync Now.

A partir de ahora, podemos utilizar la librería material design de material.io en nuestro proyecto, no sin antes aprender a solucionar nuestros primeros conflictos. También conoceremos Android X, que no es otra cosa que la versión mejorada de las librerías de soporte tradicionales, disponible a partir de la API 28.

Para evitar el error **Manifest merger failed** luego de importar material, dado que está construida basada en AndroidX, es necesario migrar a AndroidX nuestro proyecto de la siguiente forma:

1. En el menú superior izquierdo Refactor -> Migrate to AndroidX. Obviar check de respaldo a zip file.
2. Abrir el archivo `gradle.properties` en nuestra carpeta raíz e ingresar los siguientes parámetros:

```
android.useAndroidX=true  
android.enableJetifier=true
```

Estos parámetros sólo especifican que estamos usando AndroidX al Gradle.

3. Por último aprenderemos a sustituir el uso de una librería en nuestras clases java editando el archivo MainActivity.java, ubicado en la ruta /app/java la siguiente línea de código:

```
import android.support.v7.app.AppCompatActivity;
```

4. Por la siguiente:

```
import androidx.appcompat.app.AppCompatActivity;
```

5. Menú superior izquierdo Build -> Make project.
6. Para que no se nos olvide, como ya el IDE nos generó una vista `activity_main.xml`, con este cambio hacia AndroidX se nos hace necesario actualizar los elementos de la vista si no han sido actualizados con los pasos anteriores, por lo tanto, abrimos el archivo `activity_main.xml` y reemplazamos el siguiente código:

```
<android.support.constraint.ConstraintLayout
```

por el siguiente:

```
<androidx.constraintlayout.widget.ConstraintLayout
```

7. Si se nos muestra un error en el menú inferior, pestaña build output del siguiente tipo:

```
org.gradle.execution.MultipleBuildFailures: Build completed with 1 failures.
```

1. Debemos ir al menú principal seleccionando File -> Settings.
2. Para luego seleccionar Build, execution, deployment -> Instant Run
3. Desactivar todas las opciones como lo muestra la imagen 27.

Esta acción se realiza por incompatibilidad entre la funcionalidad en tiempo de ejecución instantánea (Instant Run) y básicamente librerías en lenguaje java que estamos usando.

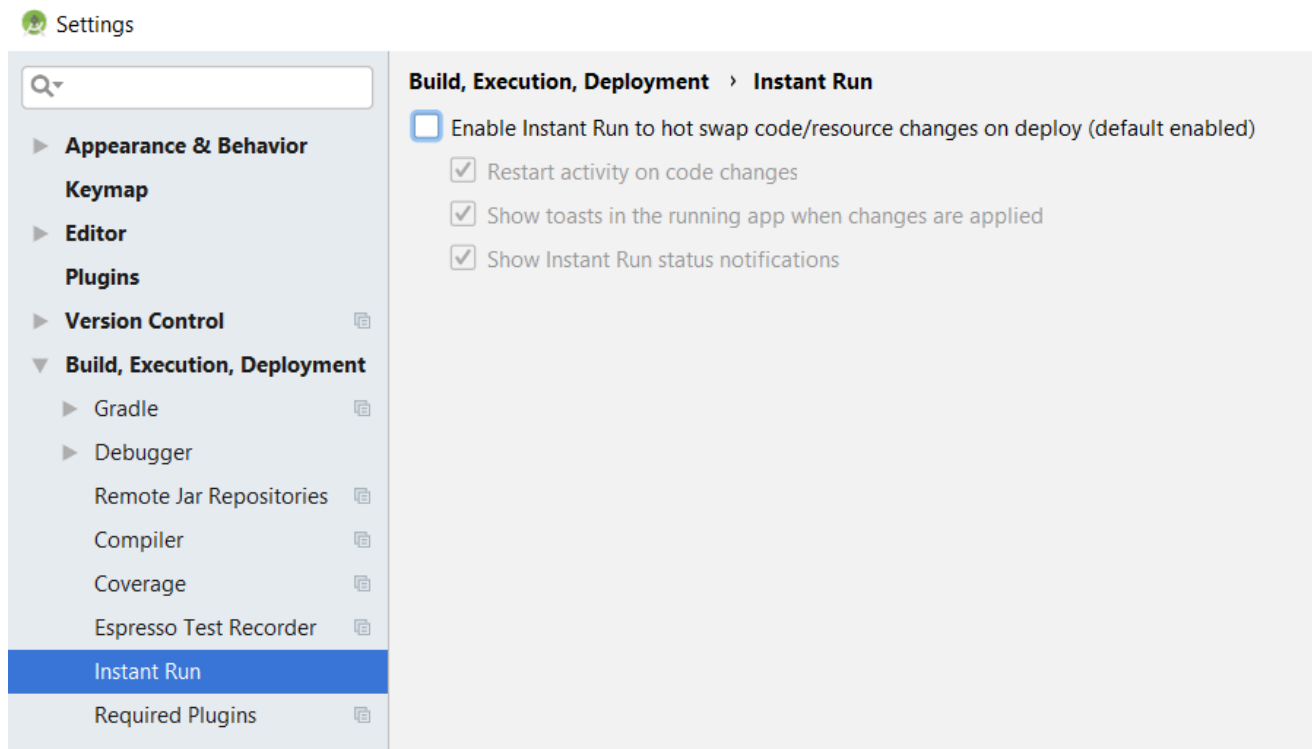


Imagen 27. Desactivar Instant Run.

La clase R

La clase R no es más que una clase auto generada por nuestro sistema operativo Android en tiempo de compilación que contiene variables constantes de tipo final, con el fin de otorgar un identificador en memoria de todos los recursos asociados a nuestras vistas ubicadas en la carpeta `/res`.

Así se ve una clase R en java:

```
/* AUTO-GENERATED FILE. DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found. It
 * should not be modified by hand.
 */

package cl.desafiolatam.android;

public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int ic_launcher=0xf020000;
    }
    public static final class id {
        public static final int menu_settings=0xf070000;
    }
    public static final class layout {
        public static final int activity_main=0xf030000;
    }
}
```


Componentes de Android

Competencias:

- Identificar componentes base de las aplicaciones Android
- Crear una activity dentro de nuestro proyecto
- Declarar una actividad en el archivo Manifest.xml
- Navegación entre actividades

Introducción

Los componentes de la aplicación son bloques de creación esenciales de una aplicación para Android. Cada componente es un punto diferente a través del cual el sistema puede ingresar a tu aplicación. No todos los componentes son puntos de entrada reales para el usuario y algunos son dependientes entre sí, pero cada uno existe como entidad individual y cumple un rol específico; cada uno es un bloque de creación único que ayuda a definir el comportamiento general de tu aplicación.

- Una componente de Actividad, básicamente agrupa todas nuestras vistas, es decir, nos muestra los textos, las imágenes, los botones menú, y enlaza sus acciones.
- Un componente Broadcast Receiver, es utilizado para identificar estados principalmente de componentes de hardware, como por ejemplo saber si tenemos o no internet en determinado momento.
- Un componente Content Provider, se utiliza en un alto nivel de desarrollo, para crear un repositorio de datos que queremos compartir con otras aplicaciones o sistemas.
- Un componente Service, se usa para manejar transacciones de datos a través de la red, interactuar con un content provider, escuchar música, todas aquellas operaciones que deban permanecer en segundo plano para no consumir recursos de memoria de la aplicación.

Componente - Activity

Una actividad representa una pantalla con interfaz de usuario. Por ejemplo, una aplicación de correo electrónico tiene una actividad que muestra una lista de los correos electrónicos nuevos, otra actividad para redactar el correo electrónico y otra actividad para leer correos electrónicos. Si bien las actividades trabajan juntas para proporcionar una experiencia de usuario consistente en la aplicación de correo electrónico, cada una es independiente de las demás. De esta manera, una aplicación diferente puede iniciar cualquiera de estas actividades (si la aplicación de correo electrónico lo permite). Por ejemplo, una aplicación de cámara puede iniciar la actividad en la aplicación de correo electrónico que redacta el nuevo mensaje para que el usuario comparta una imagen.

Una actividad se implementa como una subclase de Activity y puedes obtener más información acerca de este tema en la guía para desarrolladores Actividades.

Ejercicio 3: Creación de una segunda actividad en nuestro proyecto

MiPrimerProyecto

1. Nos ubicamos en la carpeta `/app/src/main/java/cl.desafiolatam.android` desde nuestra vista de navegación del proyecto, hacemos click derecho seleccionando, New -> Java Class , escogiendo un nombre de clase como “HomeActivity” por ejemplo y AppCompatActivity de super clase, como lo muestra la imagen 28, este proceso resultará en un nuevo archivo de clase con el código siguiente:

```
package cl.desafiolatam.android;

import androidx.appcompat.app.AppCompatActivity;

public class HomeActivity extends AppCompatActivity {
}
```

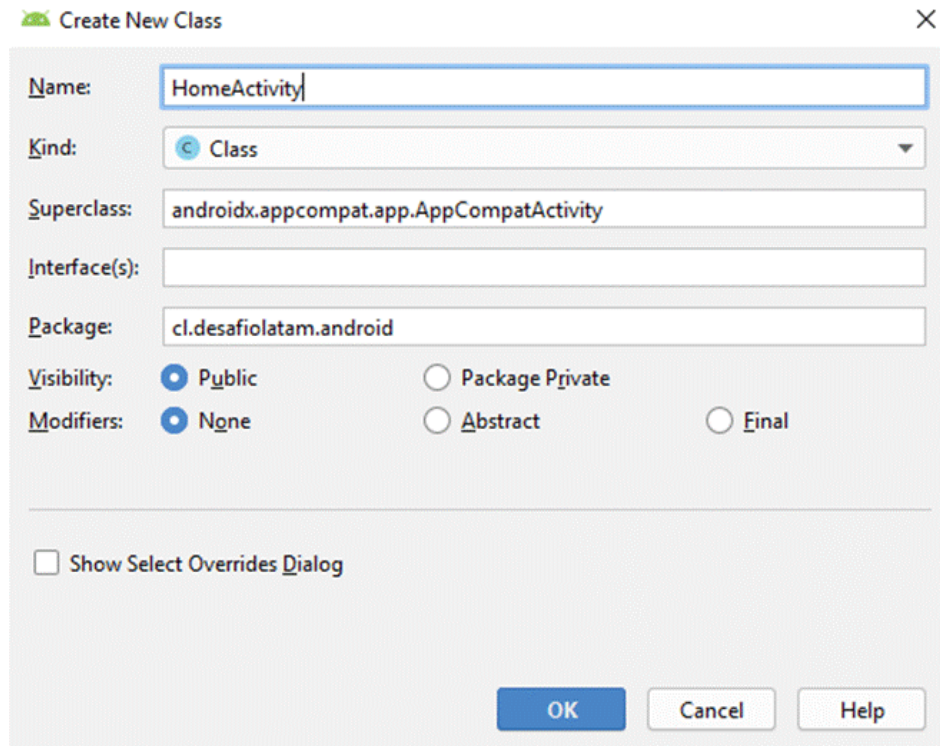


Imagen 28. Crear class HomeActivity.

Nota: Más adelante continuaremos con el desarrollo de esta actividad.

Componente - Service

Un servicio es un componente que se ejecuta en segundo plano para realizar operaciones prolongadas o tareas para procesos remotos. Un servicio no proporciona una interfaz de usuario. Por ejemplo, un servicio podría reproducir música en segundo plano mientras el usuario se encuentra en otra aplicación, o podría capturar datos en la red sin bloquear la interacción del usuario con una actividad. Otro componente, como una actividad, puede iniciar el servicio y permitir que se ejecute o enlazarse a él para interactuar.

Un servicio se implementa como una subclase de Service y puedes obtener más información acerca de este tema en la guía para desarrolladores Servicios.

Componente - Content Provider

Un proveedor de contenido administra un conjunto compartido de datos de la app. Puedes almacenar los datos en el sistema de archivos, en una base de datos SQLite, en la Web o en cualquier otra ubicación de almacenamiento persistente a la que tu aplicación pueda acceder. A través del proveedor de contenido, otras aplicaciones pueden consultar o incluso modificar los datos (si el proveedor de contenido lo permite). Por ejemplo, el sistema Android proporciona un proveedor de contenido que administra la información de contacto del usuario. De esta manera, cualquier app con los permisos correspondientes puede consultar parte del proveedor de contenido (como `ContactsContract.Data`) para la lectura y escritura de información sobre una persona específica.

Los proveedores de contenido también son útiles para leer y escribir datos privados de tu aplicación y que no se comparten. Por ejemplo, la aplicación de ejemplo Bloc de notas, usa un proveedor de contenido para guardar notas.

Un proveedor de contenido se implementa como una subclase de `ContentProvider` y debe implementar un conjunto estándar de API que permitan a otras aplicaciones realizar transacciones. Para obtener más información, lee la guía para desarrolladores Proveedores de contenido .

Componente - BroadcastReceiver

Un receptor de mensajes es un componente que responde a los anuncios de mensajes en todo el sistema. Muchos mensajes son originados por el sistema; por ejemplo, un mensaje que anuncie que se apagó la pantalla, que la batería tiene poca carga o que se tomó una foto. Las aplicaciones también pueden iniciar mensajes; por ejemplo, para permitir que otras aplicaciones sepan que se descargaron datos al dispositivo y están disponibles para usarlos. Si bien los receptores de mensajes no exhiben una interfaz de usuario, pueden crear una notificación de la barra de estado para alertar al usuario cuando se produzca un evento de mensaje. Aunque, comúnmente, un receptor de mensajes es simplemente una "puerta de enlace" a otros componentes y está destinado a realizar una cantidad mínima de trabajo. Por ejemplo, podría iniciar un servicio para que realice algunas tareas en función del evento.

Un receptor de mensajes se implementa como una subclase de BroadcastReceiver y cada receptor de mensajes se proporciona como un objeto Intent. Para obtener más información, consulta la clase BroadcastReceiver.

Iniciando componentes

Tres de los cuatro tipos de componentes (actividades, servicios y receptores de mensajes) se activan mediante un mensaje asíncrono llamado intent. Las intents enlazan componentes individuales en tiempo de ejecución (son como mensajeros que solicitan una acción de otros componentes), ya sea que el componente le pertenezca a tu aplicación o a otra.

Una intent se crea con un objeto Intent, que define un mensaje para activar un componente específico o un tipo específico de componente; una intent puede ser explícita o implícita, respectivamente.

Ejercicio 4: Activando la recién creada HomeActivity

1. Abrimos nuestro archivo AndroidManifest.xml y agregamos las siguientes líneas de código justo después de la etiqueta `</activity>` y antes de la etiqueta `</application>`:

```
<activity android:name=".HomeActivity">
<intent-filter>
<action android:name="android.intent.action.VIEW" />
<category android:name="android.intent.category.BROWSABLE" />
<data android:scheme="http"
    android:host="www.android.desafiolatam.cl"
    android:pathPrefix="/activity-home" />
</intent-filter> </activity>
```

2. Modificamos nuestra MainActivity agregando un método de inicialización de la HomeActivity y su llamado desde el método onCreate() como lo demuestra el siguiente código:

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

        initHome();
    }

    public void initHome() {
        Intent intent = new Intent(this, HomeActivity.class);
        startActivity(intent);
    }
}
```

3. Para importar la clase Intent a nuestro código de MainActivity basta con el siguiente comando `Alt + Enter`.
4. En nuestra clase HomeActivity creamos el método `onCreate()` de la siguiente forma:

```
@Override
public void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_home);
}
```

5. Es momento de que agreguemos nuestra primera imagen al proyecto, para ello, abrimos el navegador Ir a Google.
6. Ingresamos los siguientes parámetros de búsqueda "internet connection icon".
7. Seleccionamos entre las opciones del menú superior "Imágenes".
8. En el mismo menú, seleccionamos **Herramientas -> Derechos de uso -> Etiquetadas para reutilización**. De esta manera, hemos aprendido cómo buscar contenido rápidamente que no suponga ningún inconveniente legal.
9. Descargamos alguna imagen en formato png de las disponibles y la guardaremos con el nombre de "internet-service.png" en la siguiente ruta asociada a nuestro proyecto `/AndroidStudioProjects/MiPrimerProyecto/app/src/main/res/drawable`. Esta ruta se encuentra generalmente en la carpeta `Users/"Nombre de usuario"`.

10. En nuestra pantalla de navegación del proyecto nos ubicamos en la carpeta `/res/layout` , click derecho y seleccionamos `New -> Layout` resource file , escogiendo el nombre `activity_home`, y su contenido será el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <androidx.appcompat.widget.AppCompatImageView
        android:id="@+id/imgSaludo"
        android:layout_width="0dp"
        android:layout_height="200dp"
        android:contentDescription="saludo home"
        android:src="@drawable/internet_service"
        app:layout_constraintBottom_toTopOf="@id/txtSaludo"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
    />

    <androidx.appcompat.widget.AppCompatTextView
        android:id="@+id/txtSaludo"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Está es nuestra actividad MainActivity que contiene un
botón para verificar el acceso a internet"
        android:textSize="24sp"
        android:padding="20sp"
        app:layout_constraintBottom_toTopOf="@id/btnCheckInternet"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@id/imgSaludo"
        android:gravity="center"
    />

    <androidx.appcompat.widget.AppCompatButton
        android:id="@+id/btnCheckInternet"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="VERIFICAR CONEXION"
```

```
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toBottomOf="@id/txtSaludo"
android:onClick="openActivityCheckConnection"
android:background="@color/colorPrimary"
android:textColor="#FFFFFF"
android:textSize="24dp" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

De esta forma podemos visualizar al ejecutar nuestra aplicación la nueva actividad como lo muestra la imagen 29. Nótese que hemos destacado del código anterior el cómo importamos la imagen descargada "internet_service" a nuestra View ImageView.

Nota: Conoceremos más sobre las vistas cuando revisemos Views y Widgets.

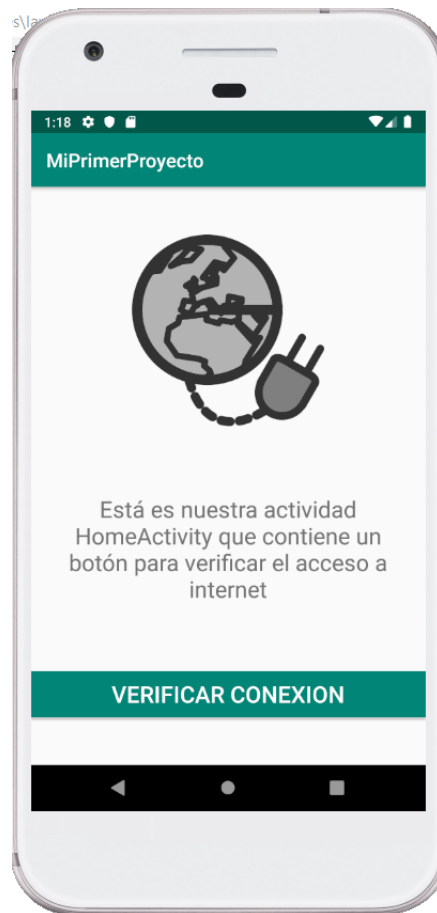


Imagen 29. Despliegue de la Actividad.

El archivo Manifest

Competencias:

- Entender la estructura que compone a un archivo manifest en Android
- Conocer las funciones del archivo Manifest

Introducción

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.

Estructura del archivo

El archivo AndroidManifest.xml tiene una estructura xml declarativa de contenido de configuración principal, entre los que destacan en su uso habitual el “uses-permission”, para declarar permisos del teléfono como acceso a internet o cámara, y la declarativa común de una `activity`. En el siguiente código se puede observar todas las posibles etiquetas de configuración en un archivo manifest:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest>
    <uses-permission />
    <permission />
    <permission-tree />
    <permission-group />
    <instrumentation />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />
    <compatible-screens />
    <supports-gl-texture />

    <application>
        <activity>
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>

        <activity-alias>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </activity-alias>

        <service>
            <intent-filter> . . . </intent-filter>
            <meta-data/>
        </service>

        <receiver>
```

```
        <intent-filter> . . . </intent-filter>
        <meta-data />
    </receiver>

    <provider>
        <grant-uri-permission />
        <meta-data />
        <path-permission />
    </provider>

    <uses-library />

</application>
</manifest>
```

Funciones del Manifest

Entre otras cosas, el archivo de manifiesto hace lo siguiente:

- Nombra el paquete de Java para la aplicación. El nombre del paquete sirve como un identificador único para la aplicación.
- Describe los componentes de la aplicación, como las actividades, los servicios, los receptores de mensajes y los proveedores de contenido que la integran. También nombra las clases que implementa cada uno de los componentes y publica sus capacidades, como los mensajes Intent con los que pueden funcionar. Estas declaraciones notifican al sistema Android los componentes y las condiciones para el lanzamiento.
- Determina los procesos que alojan a los componentes de la aplicación.
- Declara los permisos que debe tener la aplicación para acceder a las partes protegidas de una API e interactuar con otras aplicaciones. También declara los permisos que otros deben tener para interactuar con los componentes de la aplicación. Enumera las clases Instrumentation que proporcionan un perfil y otra información mientras la aplicación se ejecuta. Estas declaraciones están en el manifiesto solo mientras la aplicación se desarrolla y se quitan antes de la publicación de esta.
- Declara el nivel mínimo de Android API que requiere la aplicación.
- Enumera las bibliotecas con las que debe estar vinculada la aplicación.

Actividades

Competencias:

- Profundizar el contenido sobre las actividades y su ciclo de vida
- Crear actividades más complejas
- Crear un componente BroadcastReceiver
- Configuración de actividades en el Manifest
- Integración entre componentes de Activity y BroadcastReceiver

Introducción

Una aplicación generalmente consiste en múltiples actividades vinculadas de forma flexible entre sí. Normalmente, una actividad en una aplicación se especifica como la *actividad principal* que se presenta al usuario cuando este inicia la aplicación por primera vez. Cada actividad puede a su vez iniciar otra actividad para poder realizar diferentes acciones. Cada vez que se inicia una actividad nueva, se detiene la actividad anterior, pero el sistema conserva la actividad en una pila (la "pila de actividades").

Entendiendo las actividades en Android

Las actividades en Android es el componente principal de una aplicación en Android; es también esa interfaz que permite interactuar con el usuario a través de la pantalla y esta se ejecuta en el hilo principal del sistema operativo (UI thread).

Es una buena práctica construir un número limitado de actividades en nuestra aplicación dada la cantidad de recursos que utilizan en el sistema, su función principal tiene que ser la de contener múltiples vistas desarrolladas como fragmentos, modals, alertas, mensajes, webviews, entre otros, asegurándonos de que cada actividad sea una sección principal en nuestra aplicación.

Ejemplo de actividades de una aplicación

- LoginActivity
- HomeActivity
- HomeDetailActivity

Ciclo de vida de una Actividad

Cuando iniciamos una aplicación en nuestro dispositivo se muestra una actividad. A medida que vamos navegando por la aplicación esta actividad pasa por distintos estados entre su creación y eliminación, existen estados intermedios de inicio, pausa, resumen, detención que se activan cuando realizamos acciones o abrimos otras aplicaciones. En la imagen 30 podemos visualizar el ciclo oficial de una actividad en Android.

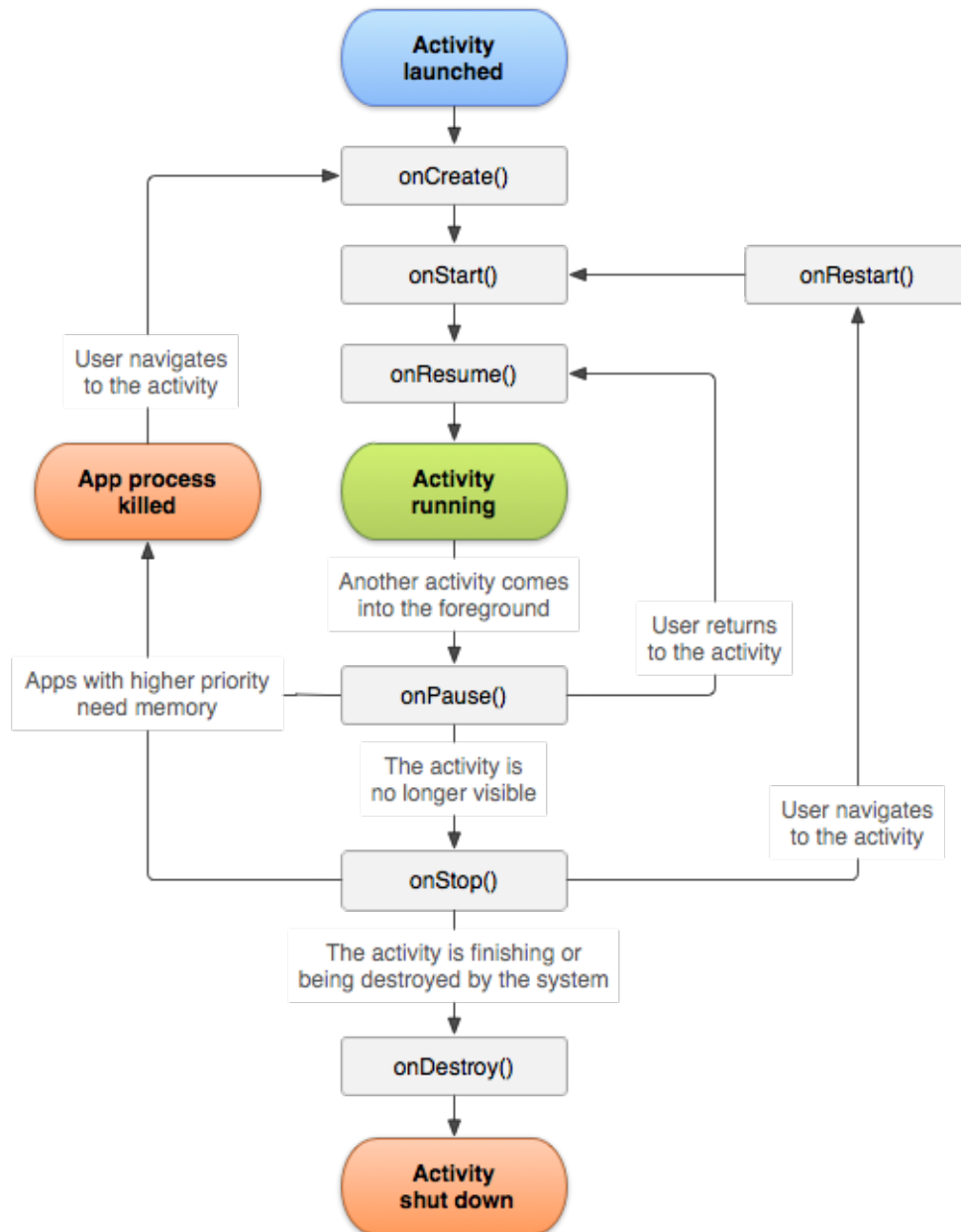


Imagen 30: Ciclo de vida del las actividades.

Configuración de actividad en el archivo Manifest

Toda actividad debe ser declarada en el archivo AndroidManifest.xml de nuestra carpeta raíz. Recordemos el ejercicio realizado anteriormente, en el cual creamos una nueva actividad con el siguiente ejercicio:

Ejercicio 5: Agregar una tercera actividad llamada HomeDetailActivity a nuestro archivo manifest.

1. Abrimos nuestro archivo AndroidManifest.xml
2. Agregamos la nueva actividad después de la última etiqueta de cierre `</activity>` y antes de la etiqueta de cierre `</manifest>` de la siguiente forma:

```
<activity android:name=".HomeDetailActivity">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.BROWSABLE" />
        <data android:scheme="http"
            android:host="www.android.desafiolatam.cl"
            android:pathPrefix="/activity-homedetail" />
    </intent-filter>
</activity>
```

3. Agreguemos permisos de uso de internet y reconocimiento de estados de conexión en el archivo antes de la etiqueta de apertura `<application>` con el siguiente código:

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

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

4. Declaramos un receiver, para usar el componente BroadcastReceiver antes de la etiqueta de cierre `</application>` de la siguiente manera:

```
<receiver android:name=".utils.NetworkStatusReceiver">
    <intent-filter>
        <action android:name="android.net.conn.CONNECTIVITY_CHANGE" />
    </intent-filter>
</receiver>
```

Creación de una nueva actividad en el proyecto

En esta sección vamos a repasar lo visto para la creación de una actividad, aprovechando la oportunidad para profundizar el contenido del curso, demostrando un caso de la vida real que será útil para su vida laboral.

Comencemos por agregar un nuevo componente principal de tipo `Broadcast receiver`, además de declarar acciones entre distintos ciclos de vida de una actividad para aprender cómo trabajan. El fin del siguiente ejercicio será construir una actividad que verifique el estado de nuestra conexión a internet en tiempo real, desactivando y activando nuestra wifi y datos como prueba. Esta nueva actividad será llamada desde la anterior `HomeActivity` a través de un botón que vamos a agregar para realizar la acción.

Ejercicio 6: Creación de actividad `HomeDetailActivity`

1. Creamos una nueva actividad denominada `HomeDetailActivity`, donde verificaremos la disponibilidad que tiene el dispositivo para el uso de internet en tiempo real, diferenciando entre la wifi y el uso de los datos, además trabajaremos con el componente `BroadcastReceiver` a través de intents para recibir las respuestas que nos devolverá el teléfono de forma nativa desde sus drivers de conexión, como lo demuestra el siguiente código:

```
public class HomeDetailActivity extends AppCompatActivity {

    private BroadcastReceiver broadcastNetworkReceiver;
    static TextView connectionStatus;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_home_detail);
        connectionStatus= findViewById(R.id.connectionStatus);
        broadcastNetworkReceiver = new NetworkStatusReceiver();
        registerNetworkBroadcastForNougat();
    }

    public static void messageConnectionStatus(boolean value){

        if(value){
            connectionStatus.setText("!Tenemos acceso a internet!");
            connectionStatus.setBackgroundColor(Color.GREEN);
            connectionStatus.setTextColor(Color.WHITE);
        }
    }
}
```

```

    }else {
        connectionStatus.setVisibility(View.VISIBLE);
        connectionStatus.setText("¡No podemos conectarnos!");
        connectionStatus.setBackgroundColor(Color.RED);
        connectionStatus.setTextColor(Color.WHITE);
    }
}

private void registerNetworkBroadcastForNougat() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.N) {
        registerReceiver(broadcastNetworkReceiver, new
IntentFilter(ConnectivityManager.CONNECTIVITY_ACTION));
    }

}

protected void unregisterNetworkChanges() {
    try {
        unregisterReceiver(broadcastNetworkReceiver);
    } catch (IllegalArgumentException e) {
        e.printStackTrace();
    }
}

@Override
public void onDestroy() {
    super.onDestroy();
    unregisterNetworkChanges();
}
}

```


2. Nuestra HomeDetailActivity necesita un archivo layout de vista para mostrar al usuario del dispositivo las nuevas funcionalidades, por lo tanto creamos el archivo `activity_home_detail.xml` en la carpeta `/res/layout`, con el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".HomeDetailActivity">

    <androidx.appcompat.widget.AppCompatTextView
        android:id="@+id/connectionStatus"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="25dp"
        android:text="Conexión establecida"
        app:layout_constraintBottom_toTopOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="parent"
        android:gravity="center" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

3. Creamos una clase para implementar el BroadcastReceiver denominada NetworkStatusReceiver para practicar el uso del componente y usar sus métodos requeridos (onReceive), en donde también haremos uso del `ConnectivityManager` y del `NetworkInfo` para verificar los estados de conexión, con el siguiente código:

```
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.net.ConnectivityManager;
import android.net.NetworkInfo;
import android.util.Log;
import static
cl.desafiolatam.android.HomeDetailActivity.messageConnectionStatus;

public class NetworkStatusReceiver extends BroadcastReceiver{

    @Override
    public void onReceive(Context context, Intent intent)
    {
        try
        {
            if (isOnline(context)) {
                messageConnectionStatus(true);
                Log.i("Info: ", "Estamos conectados a la internet desde el
dispositivo");
            } else {
                messageConnectionStatus(false);
                Log.i("Info Error: ", "Se perdió la conexión a internet");
            }
        } catch (NullPointerException e) {
            e.printStackTrace();
        }
    }

    private boolean isOnline(Context context) {
        try {
            ConnectivityManager cm = (ConnectivityManager)
context.getSystemService(Context.CONNECTIVITY_SERVICE);
            NetworkInfo netInfo = cm.getActiveNetworkInfo();
            return (netInfo != null && netInfo.isConnected());
        } catch (NullPointerException e) {
            e.printStackTrace();
            return false;
        }
    }
}
```

4. Agregamos un nuevo método a nuestra antigua clase HomeActivity que inicie nuestra nueva actividad al ser accionado de la siguiente forma:

```
public void openActivityCheckConnection(View v){  
    Intent intent = new Intent(this, HomeDetailActivity.class);  
    startActivity(intent);  
}
```

5. Editamos el archivo activity_home.xml para incorporar un botón que nos llevará a una nueva actividad que verifica nuestra conexión a internet. Prestar especial atención a la línea android:onClick del AppCompatActivity ya que esta enlaza directamente la acción con el método declarado en nuestra clase en el punto 4. Este es el código:

```
<androidx.appcompat.widget.AppCompatButton  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="VERIFICAR CONEXION"  
    app:layout_constraintBottom_toBottomOf="parent"  
    app:layout_constraintLeft_toLeftOf="parent"  
    app:layout_constraintRight_toRightOf="parent"  
    app:layout_constraintTop_toBottomOf="@id/txtSaludo"  
    android:onClick="openActivityCheckConnection"  
    android:background="@color/colorPrimary"  
    android:textColor="#FFFFFF"  
    android:textSize="24dp"/>
```

De esta forma podemos visualizar al ejecutar nuestra aplicación la nueva actividad como lo muestra la imagen 31 y 32 haciendo la verificación de la conexión a internet.



Imagen 31. Con acceso a internet.

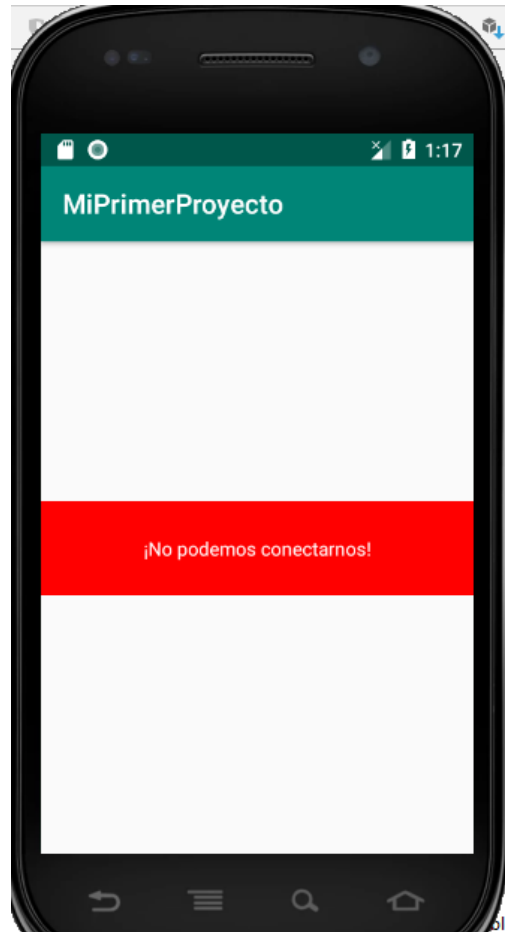


Imagen 32. Sin acceso a internet.