

SEMINARIO DE LENGUAJES

OPCIÓN ANDROID



Layouts

Esp. Delía Lisandro, Mg. Corbalán Leonardo

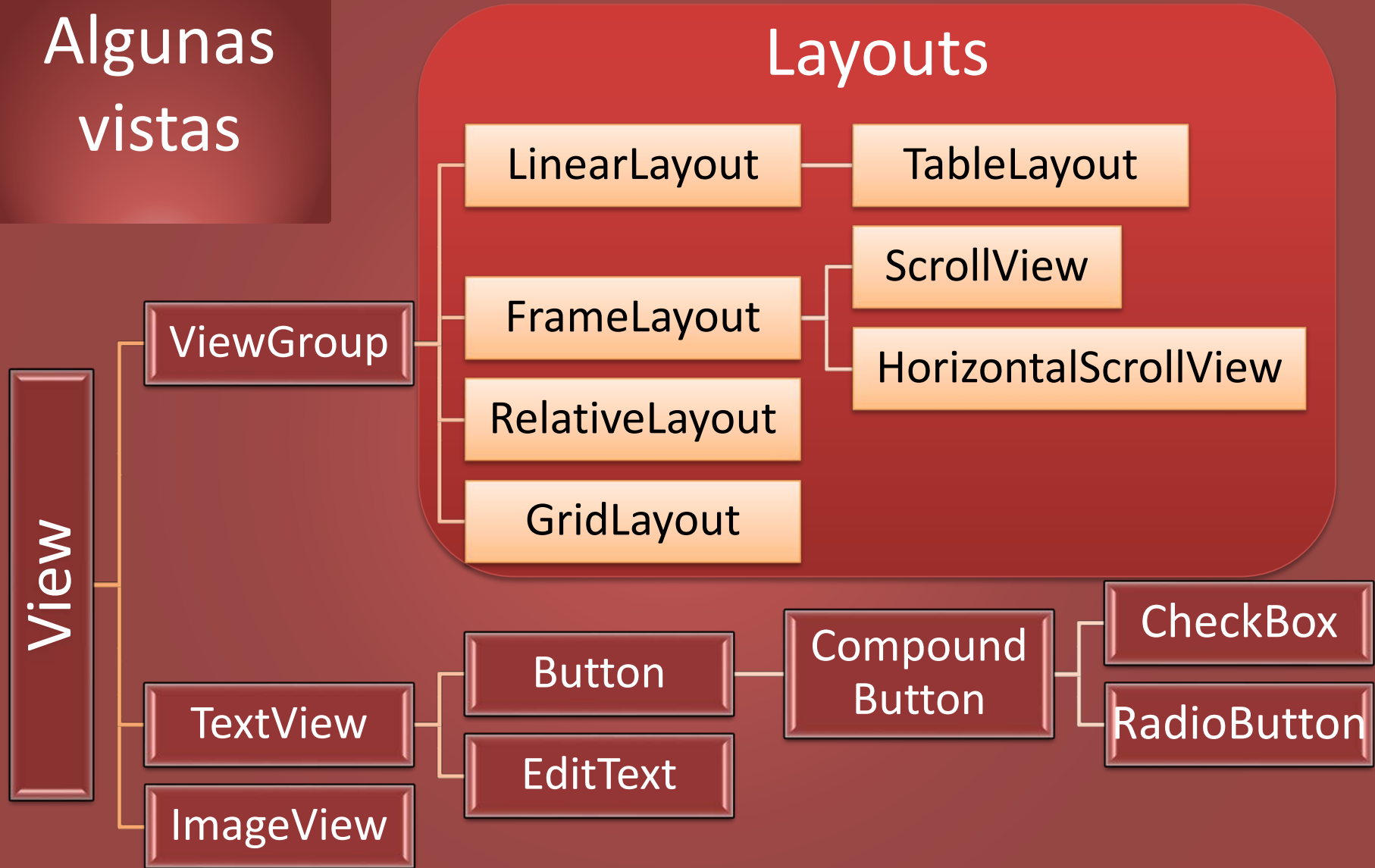
Layouts

- El **layout** de una **activity** representa el diseño de la interfaz de usuario determinando la disposición de distintos componentes visuales (vistas o **views**) en la misma.
- Los **layouts** también son vistas pero pertenecen a una categoría específica de vistas (**ViewGroup**) capaz de contener a otras vistas

Viewgroups

- Los elementos visuales simples, como el **EditText**, **TextView**, **Button**, etc. son clases particulares de vistas (**View**) que deben ser dispuestos dentro de un contenedor
- El contenedor es un **ViewGroup** que define el modo que se muestran los elementos hijos que aloja.
- Conoceremos algunos de los **viewGroups** más populares

Algunas vistas

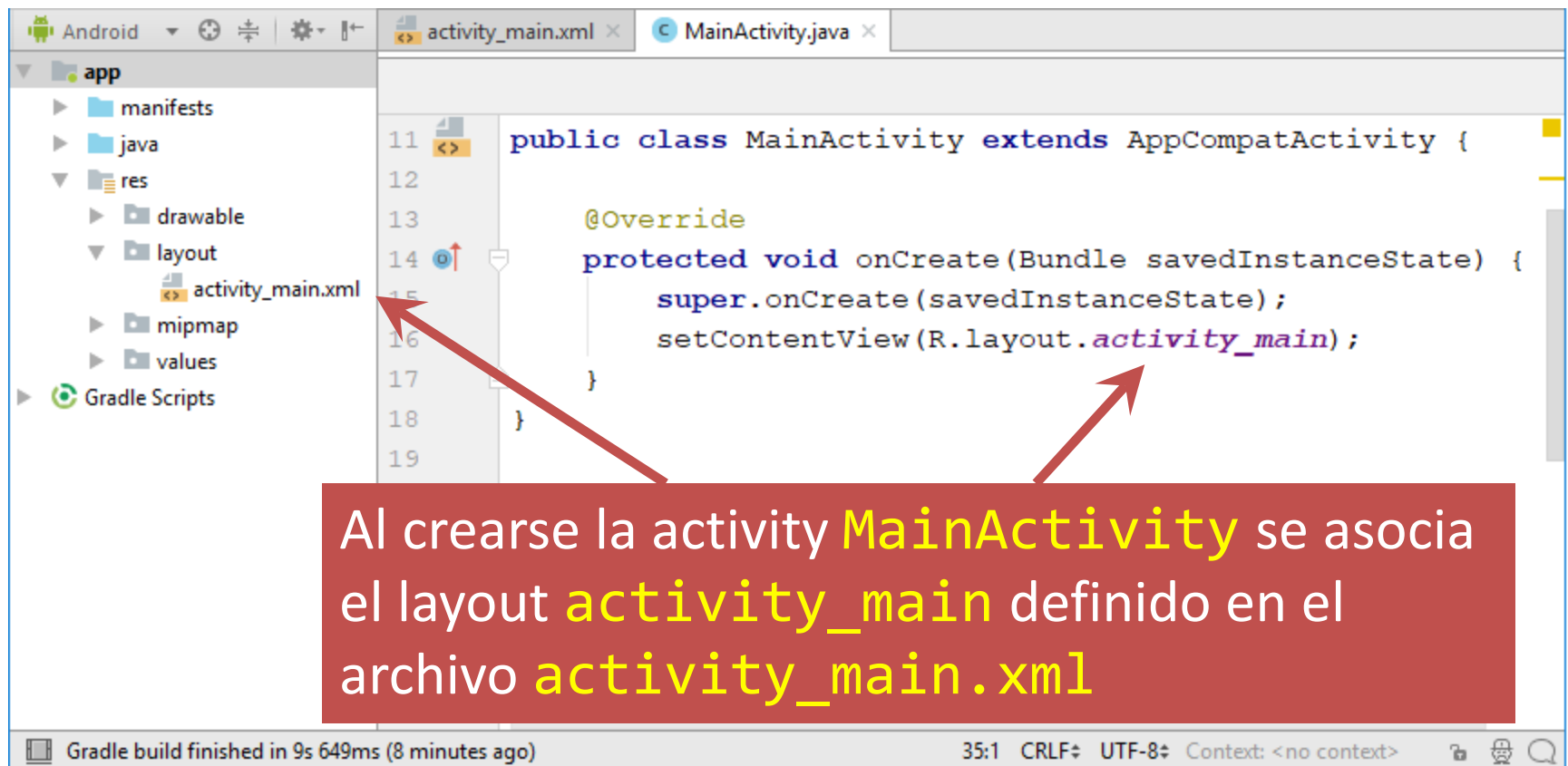


Layouts

- La interfaz de usuario puede ser definida mediante:
 - Archivos XML
 - En tiempo de ejecución, programáticamente.

Layouts desde archivos XML

- Las **Activities** que definen su interfaz por medio de **archivos XML** se asocian a estos archivos mediante **setContentView()** en el callback **onCreate()**



The screenshot shows the Android Studio IDE. On the left, the 'app' folder is expanded, showing the 'layout' subfolder where 'activity_main.xml' is located. On the right, the 'MainActivity.java' file is open, showing the following code:

```
11 public class MainActivity extends AppCompatActivity {  
12  
13     @Override  
14     protected void onCreate(Bundle savedInstanceState) {  
15         super.onCreate(savedInstanceState);  
16         setContentView(R.layout.activity_main);  
17     }  
18 }  
19
```

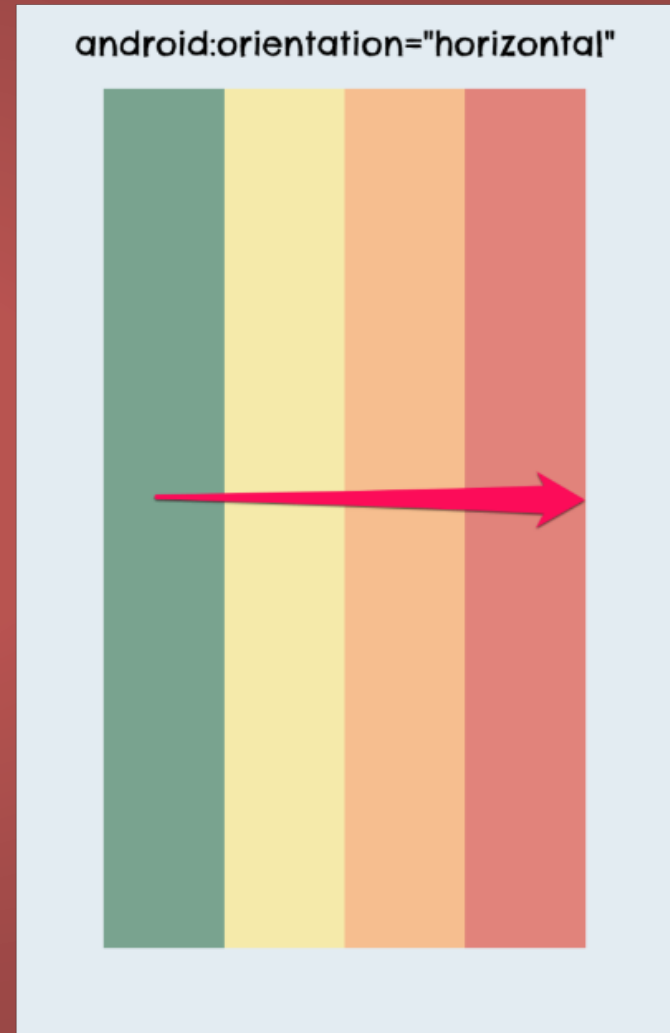
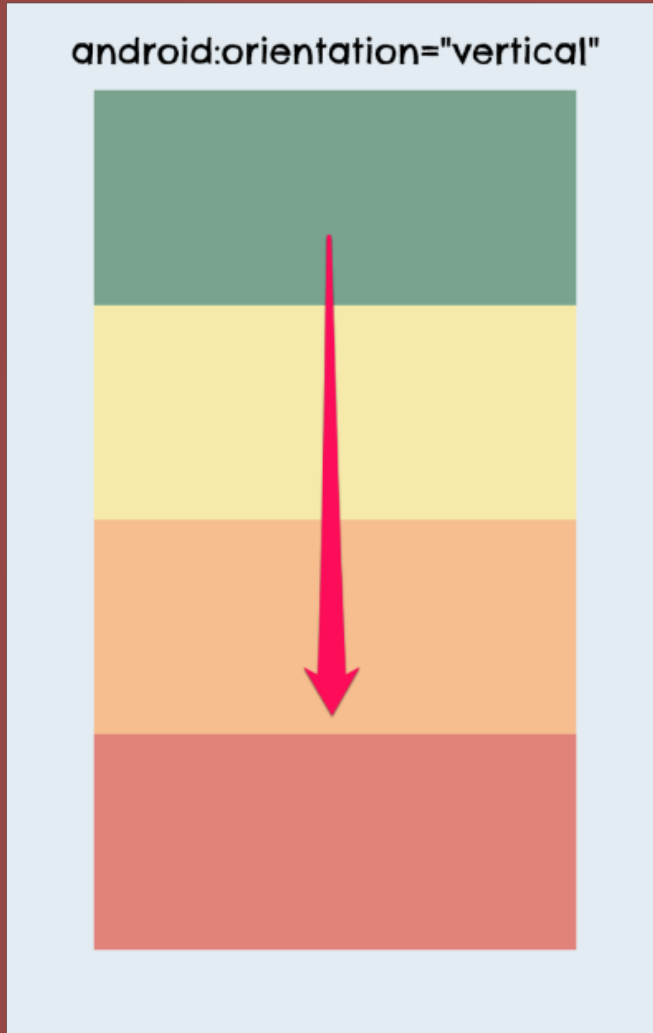
Two red arrows point from the text box below to the code: one points to the `activity_main` parameter in the `setContentView` call on line 16, and the other points to the `activity_main.xml` file in the project explorer on the left.

Al crearse la activity **MainActivity** se asocia el layout **activity_main** definido en el archivo **activity_main.xml**

Gradle build finished in 9s 649ms (8 minutes ago) 35:1 CRLF UTF-8 Context: <no context>

Vamos a presentar alguno de
los **layouts** más populares

LinearLayout

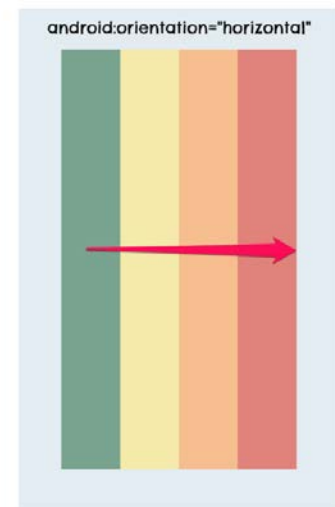
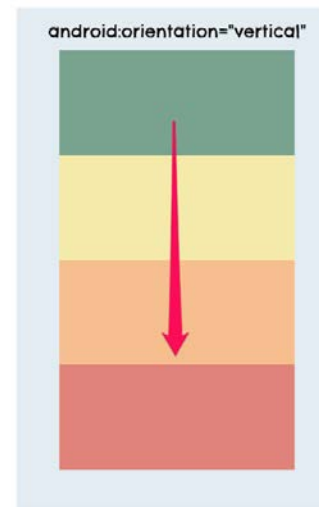


Ya lo hemos utilizado en clases anteriores

LinearLayout

- Es un **ViewGroup** que alinea a los elementos hijos en una única dirección.
- La dirección puede ser vertical u horizontal.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    ----
</LinearLayout>
```

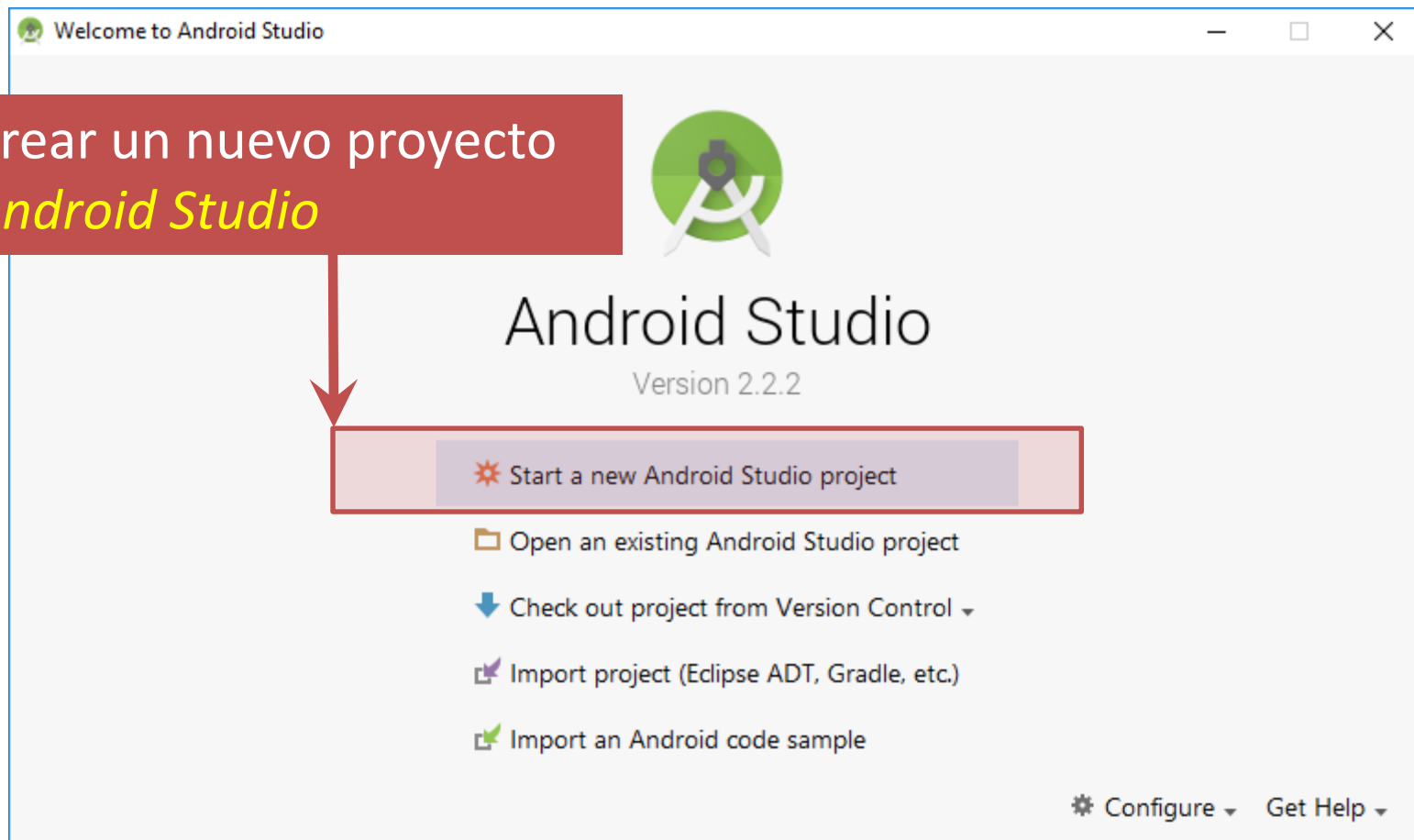


Ejercitación



Iniciar **Android Studio**
Cerrar cualquier proyecto que
pudiese estar abierto

Crear un nuevo proyecto
Android Studio





Create Android Project

Application name

TeoriaLayouts

Company domain

curso.android.com

Project location

C:\Users

Package name

com.android.curso.teorialayouts

Edit

☐ Include C++ support

☐ Include Kotlin support

Elegir los nombres de la
aplicación y del dominio

Previous

Next

Cancel

Finish



Create Android Project

Application name

TeoriaLa

Company

curso.ar

Project location

C:\Users

Package name

com.android.curso.teorialayouts

Edit

☐ Include C++ support☐ Include Kotlin support

Previous

Next

Cancel

Finish

Ambos nombres son utilizados por **Android Studio** para establecer el nombre del **package**. En caso de subir la aplicación a **Google Play Store**, el nombre del paquete será un **identificador único** para la aplicación en la tienda.



Target Android Devices

Select the form factors your app will run on

Different platforms may require separate SDKs

☒ Phone and Tablet

Minimum SDK API 19: Android 4.4 (KitKat)

Lower API levels target more devices, but have fewer features available.

By targeting API 19 and later, your app will run on approximately **73,9%** of the devices that are active on the Google Play Store.

[Help me choose](#)

☐ Wear

Minimum SDK API

☐ TV

Minimum SDK API

☐ Android Auto

☐ Glass

Minimum SDK Glass Development Kit Preview (API 19)

Elegir API 19: Android 4,4 (KitKat) como mínimo SDK requerido por la aplicación

Previous

Next

Cancel

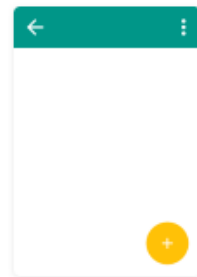
Finish



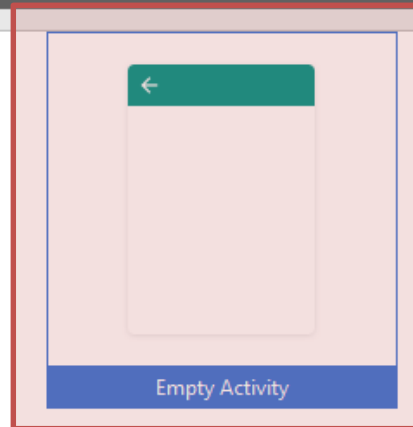
Add an Activity to Mobile



Add No Activity



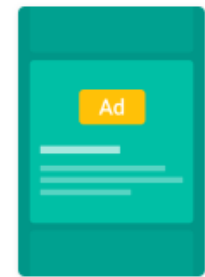
Basic Activity



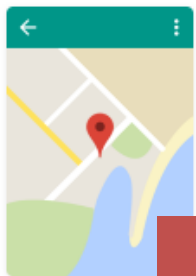
Empty Activity



Fullscreen Activity



Google AdMob Ads Activity



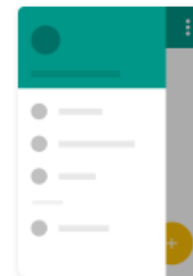
Google Maps Activity



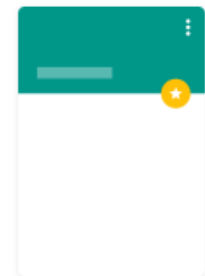
Login Activity



Master/Detail Flow



Navigation Drawer Activity



Scrolling Activity

Elegir **Empty Activity**

Previous

Next

Cancel

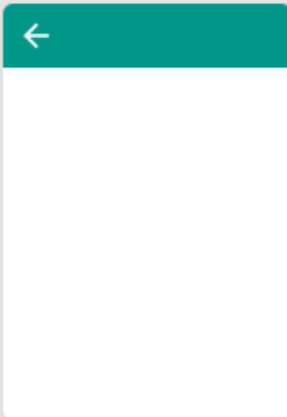
Finish



Customize the Activity



Creates a new empty activity



Activity Name: MainActivity

☒ Generate Layout File

Layout Name: activity_main

☒ Backwards Compatibility (AppCompat)

Dejar el *Activity Name* y *Layout Name* que propone Android Studio y presionar Finish

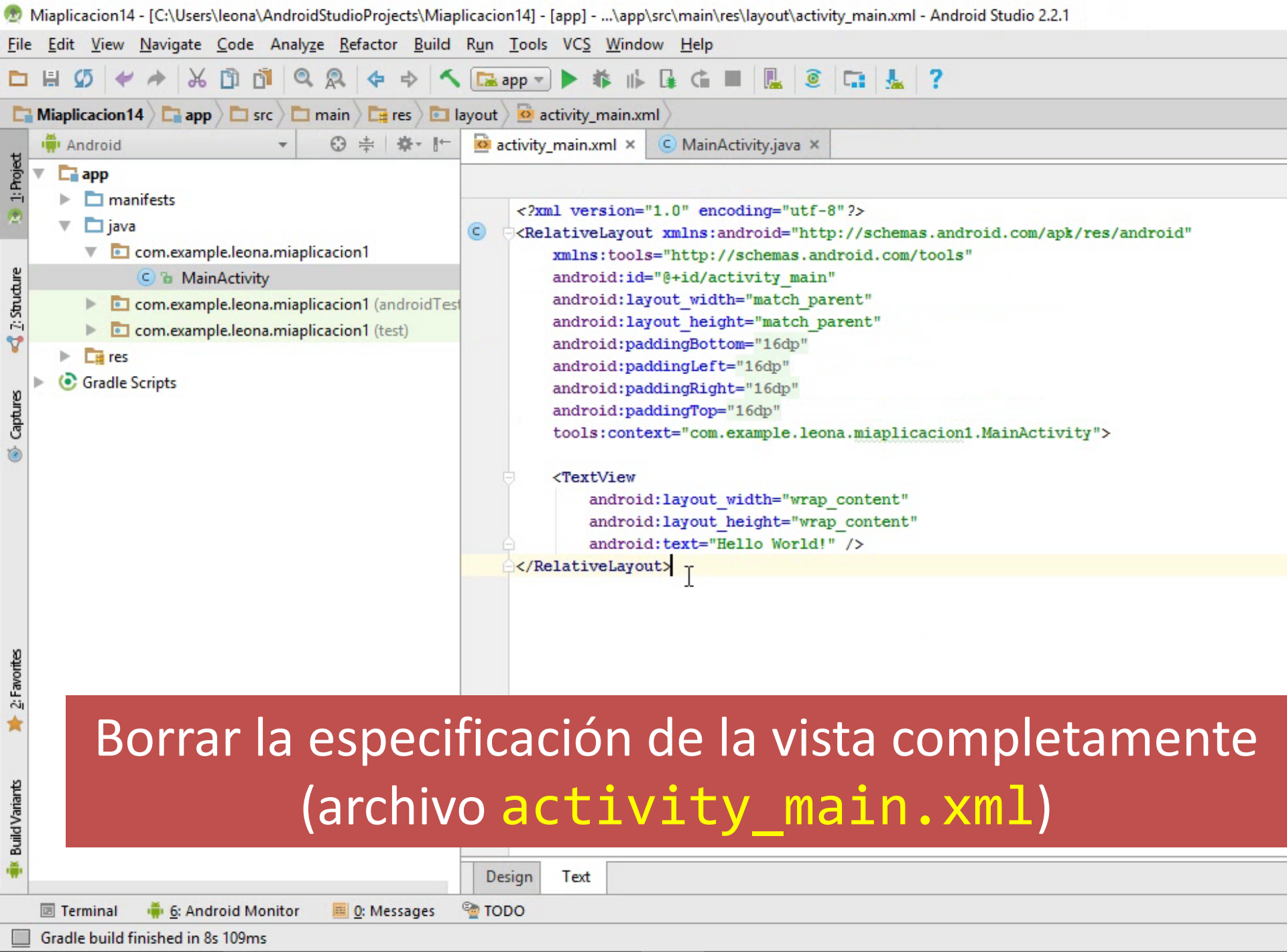
The name of the activity class to create

Previous

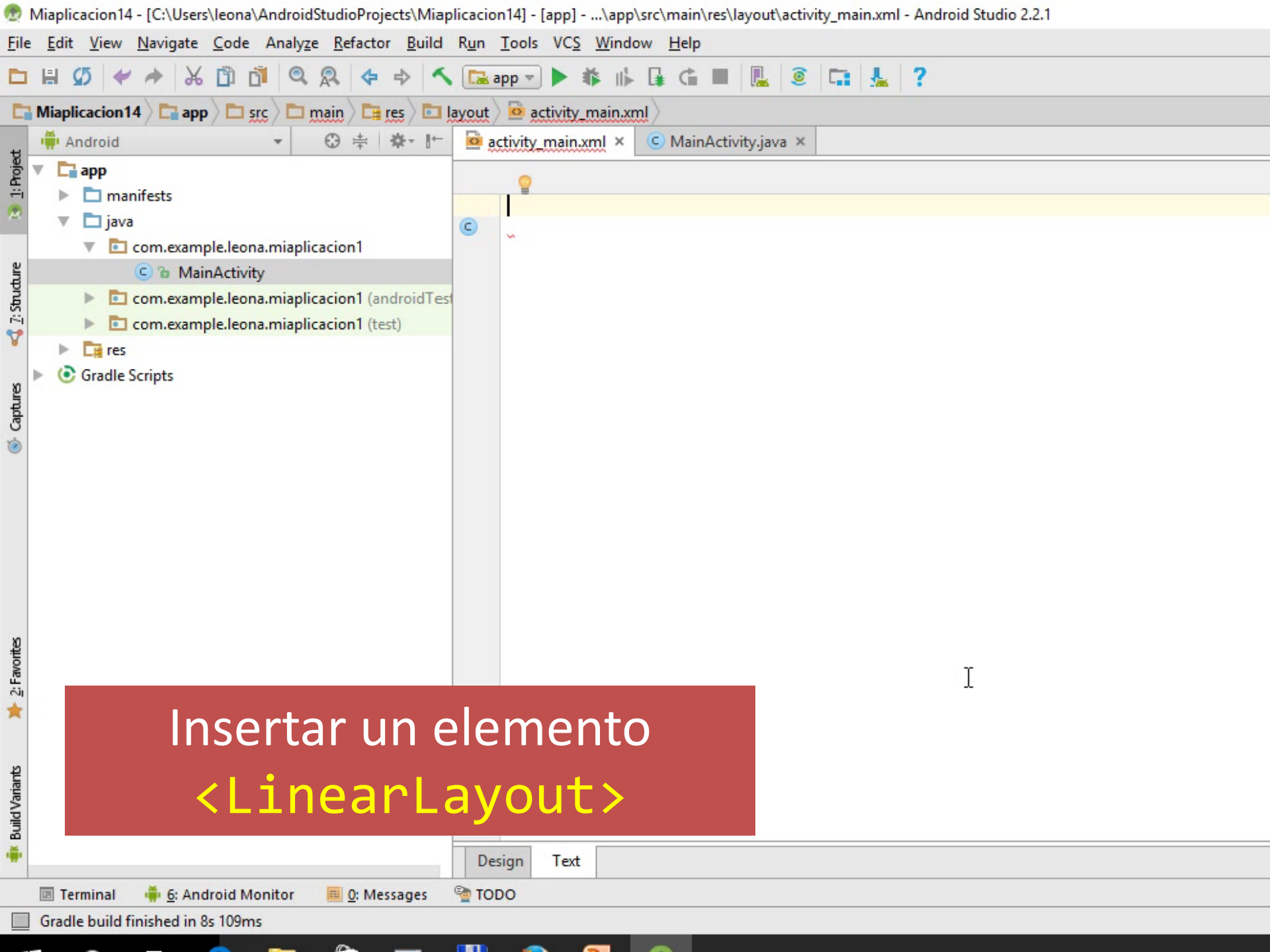
Next

Cancel

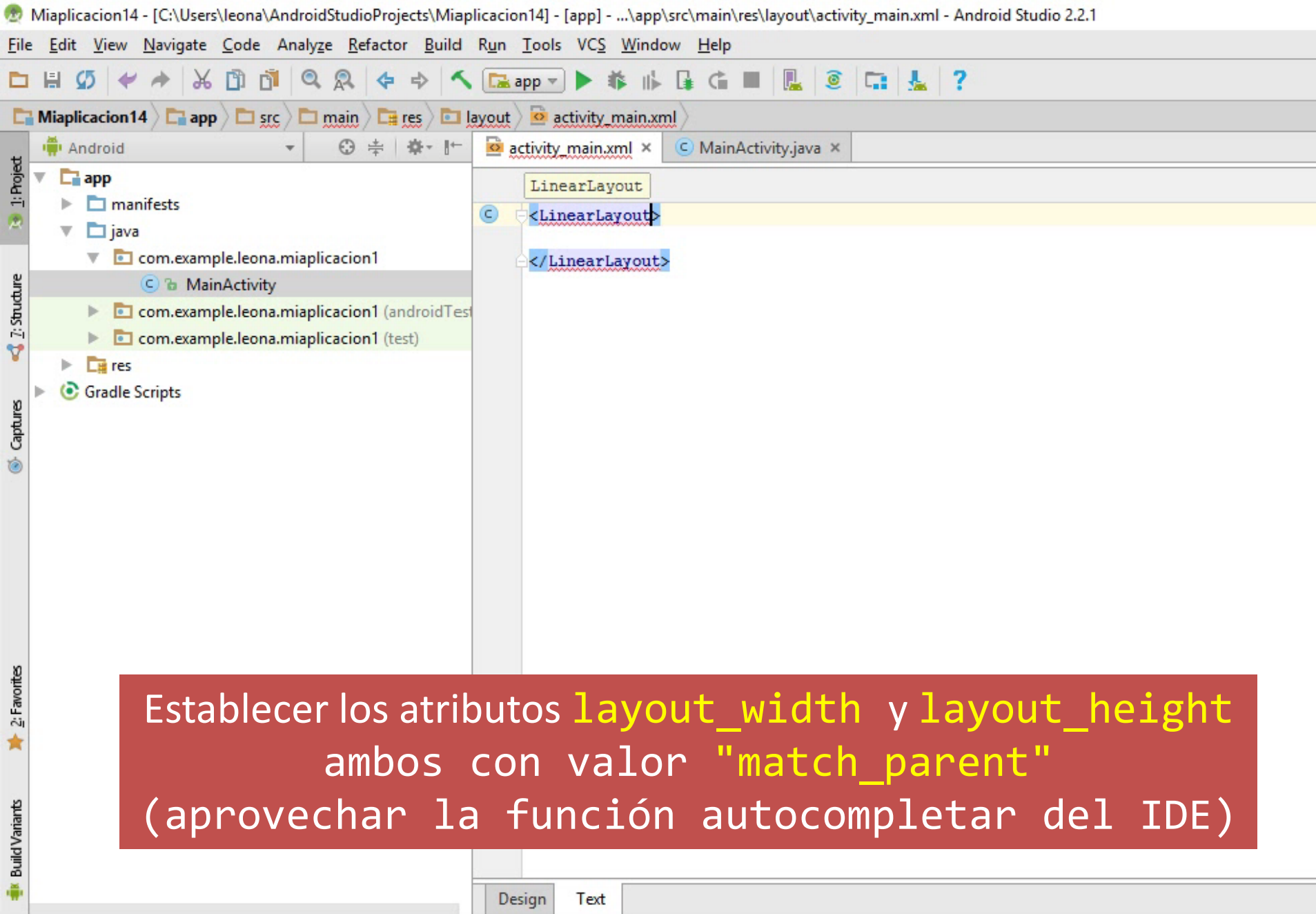
Finish



Borrar la especificación de la vista completamente
(archivo **activity_main.xml**)



Insertar un elemento
<LinearLayout>



Establecer los atributos `layout_width` y `layout_height` ambos con valor `"match_parent"` (aprovechar la función autocompletar del IDE)

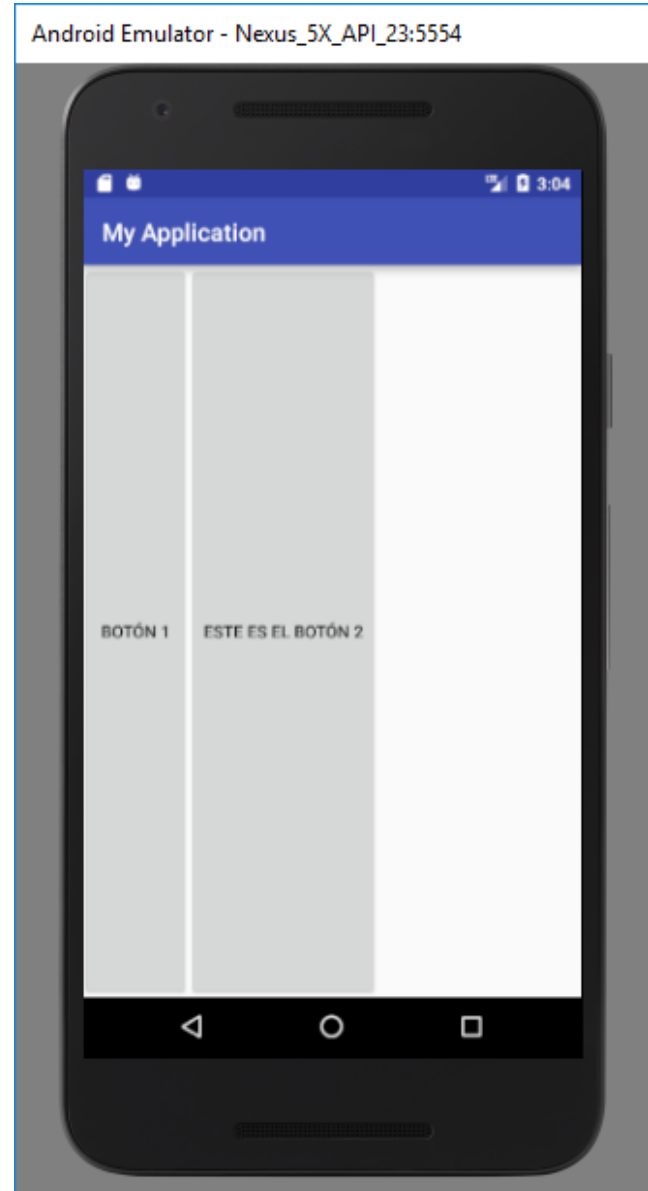
Agregar dos botones al **Layout** y ejecutar la aplicación

```
<LinearLayout android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <Button
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:text="Botón 1" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:text="Este es el botón 2"/>
</LinearLayout>
```

*¿Qué efecto tienen los valores **match_parent** y **wrap_content**?*

Orientación en LinearLayout

Por defecto la orientación de los elementos hijos se visualizan horizontalmente, de izquierda a derecha.



Orientación en LinearLayout

- Probar especificando el atributo **orientation** a modo vertical para que se vea de esta forma.
- ¿Qué valores deben tener los atributos **layout_width** y **layout_height** de los botones?

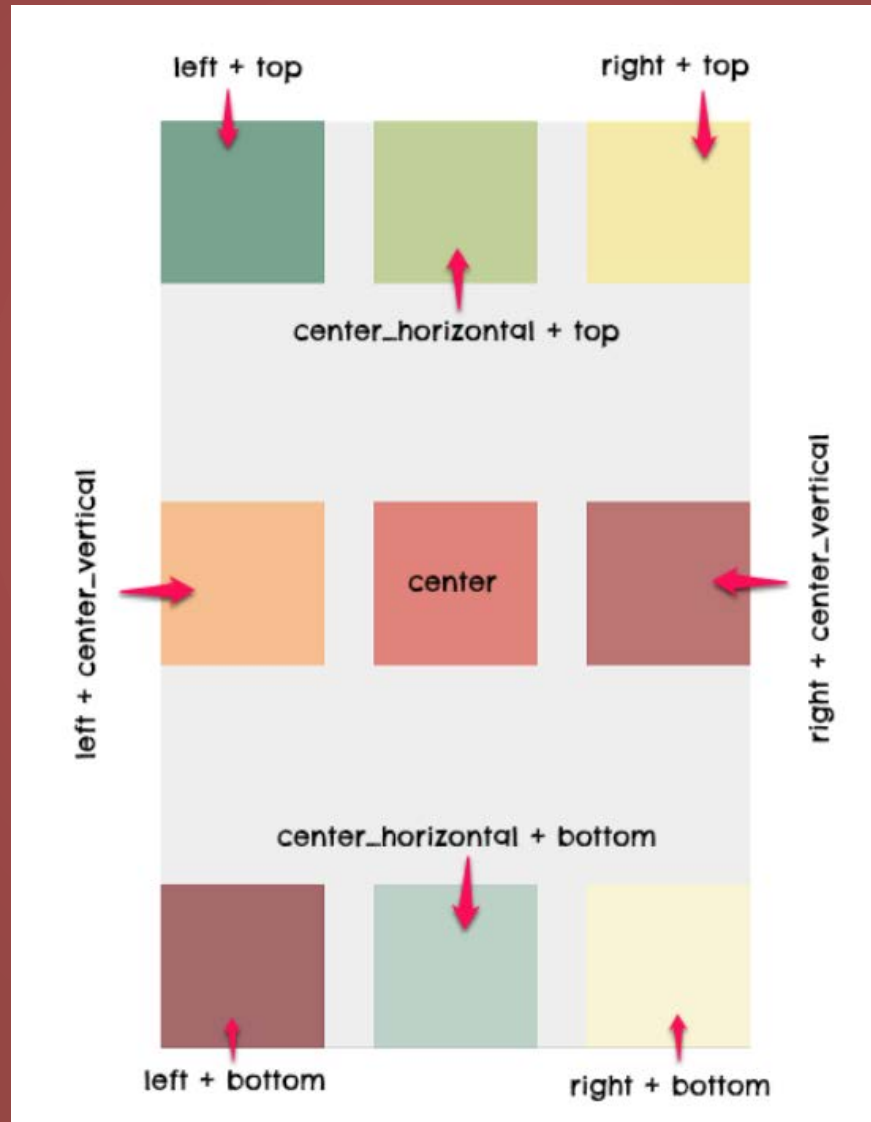


Solución

```
<LinearLayout android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical">
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Botón 1" />

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Este es el botón 2" />
</LinearLayout>
```

FrameLayout



FrameLayout

Modificar la vista de la **activity** principal de la siguiente manera

```
<FrameLayout android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="100dp"
        android:text="Primer texto"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="50dp"
        android:text="Segundo texto"
    />
</FrameLayout>
```

FrameLayout

Modificar la vista de la **activity** principal de la siguiente manera



Un **FrameLayout** es un **ViewGroup** simple y eficiente.

Pensado para ser usado con un **view** hijo o con **Views** que admitan solapamiento.

Probar y responder



¿Cuál es el efecto en las vistas de los siguiente atributos ?

`android:layout_gravity="right"`

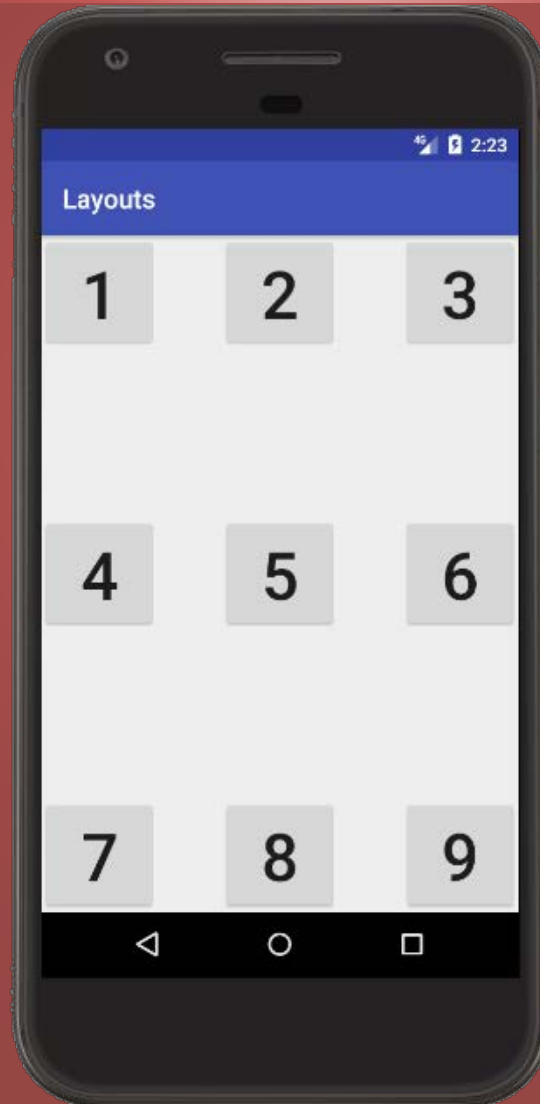
`android:layout_gravity="bottom"`

`android:layout_gravity="center"`

`android:layout_gravity="center|right"`

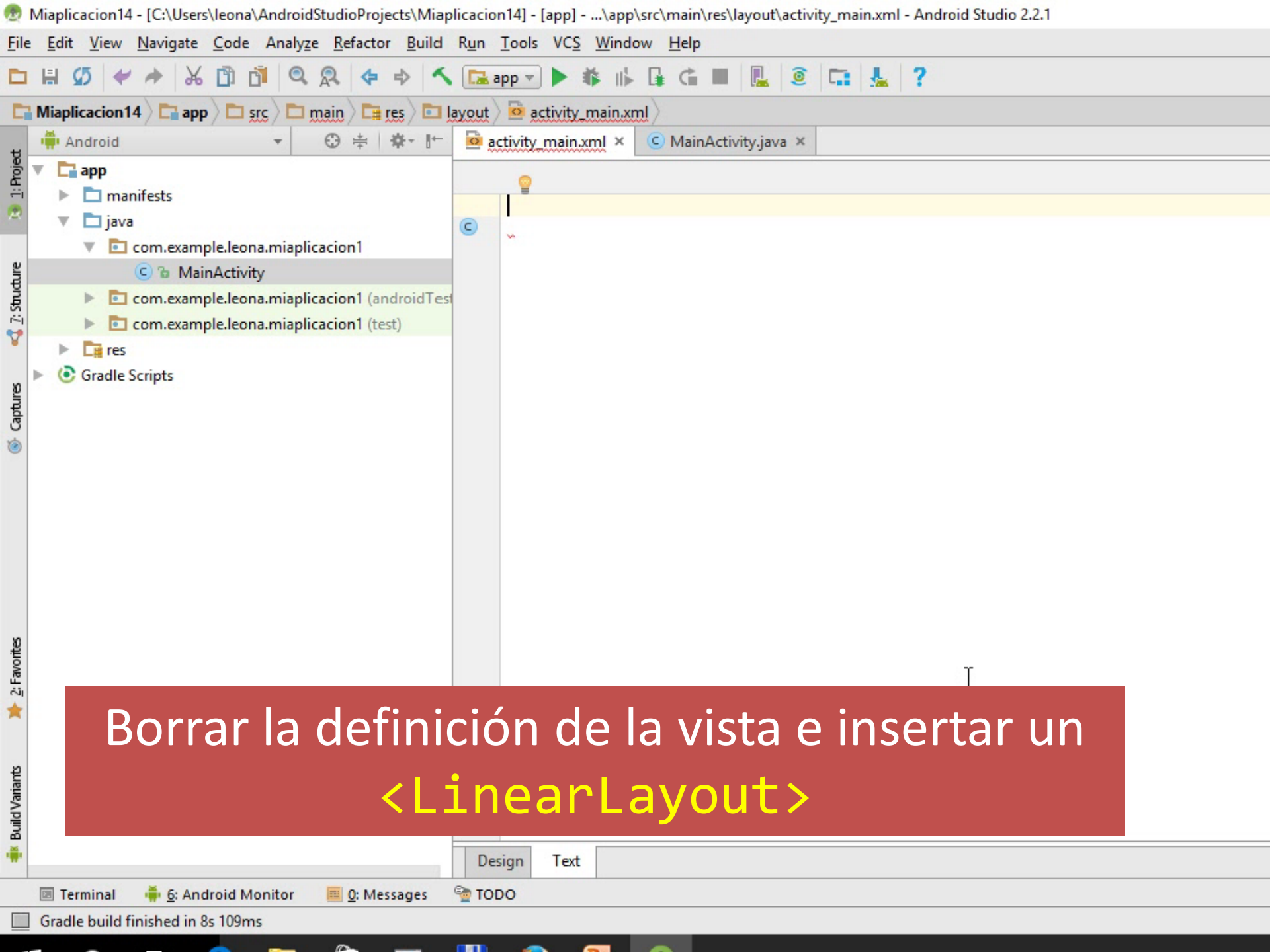
Ejercicio

Codificar la siguiente **activity**

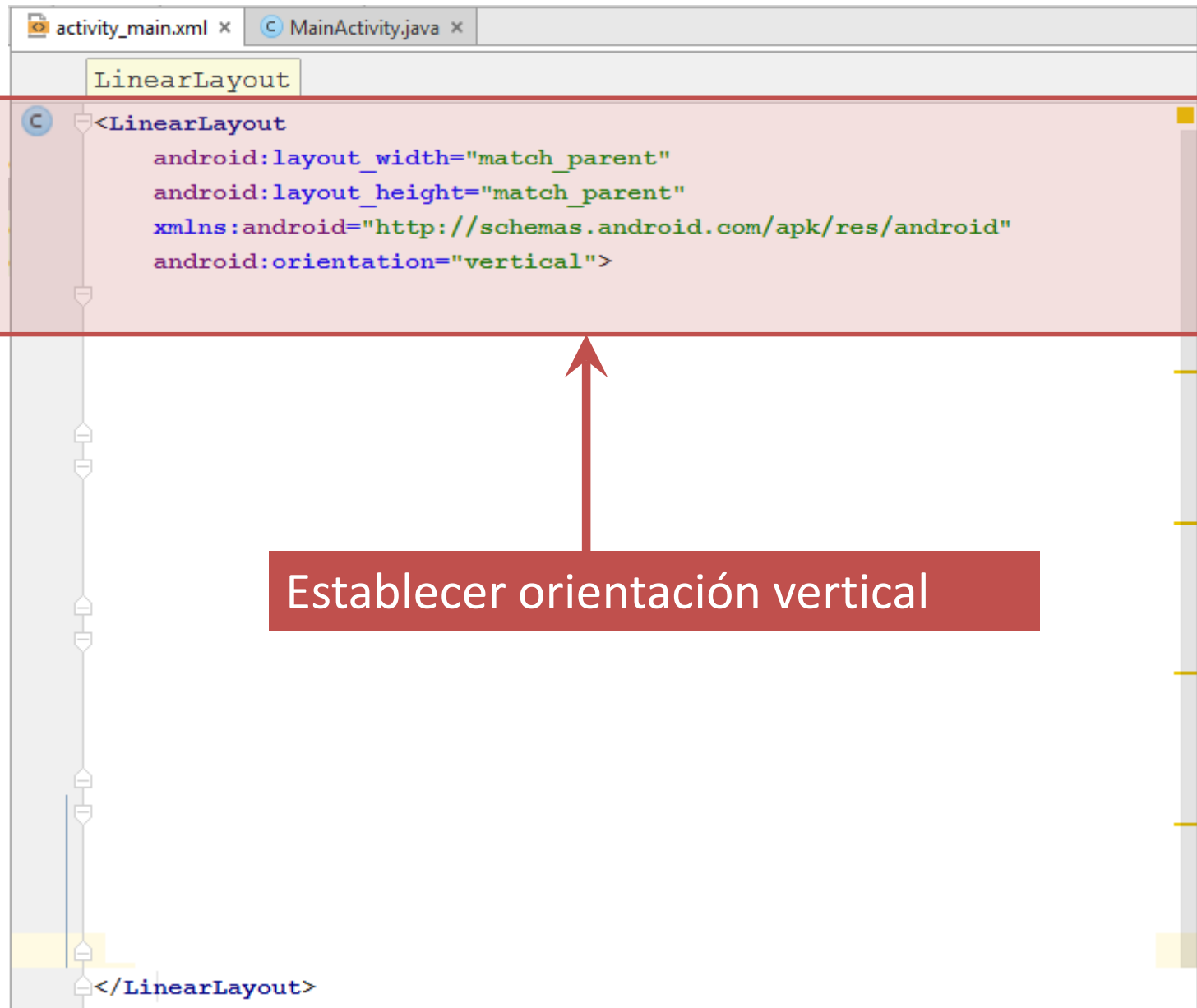


ScrollView y HorizontalScrollView

- **ScrollView** es un **FrameLayout** especializado que puede hacer **scroll vertical** sobre el elemento que contiene (sólo puede alojar un único hijo)
- **HorizontalScrollView** es un **FrameLayout** especializado que puede hacer **scroll horizontal** sobre el elemento que contiene (sólo puede alojar un único hijo)
- El ejemplo a continuación mostrará la necesidad de utilizar un **ScrollView**



Borrar la definición de la vista e insertar un
<LinearLayout>



The screenshot shows the Android Studio IDE with two tabs: 'activity_main.xml' and 'MainActivity.java'. The 'activity_main.xml' tab is active, displaying the XML layout code. The code defines a `LinearLayout` with the following attributes: `android:layout_width="match_parent"`, `android:layout_height="match_parent"`, `xmlns:android="http://schemas.android.com/apk/res/android"`, and `android:orientation="vertical"`. Inside the `LinearLayout`, there is a `Button` element with attributes: `android:layout_width="match_parent"`, `android:layout_height="150dp"`, `android:text="Botón 1"`, and `android:layout_margin="10dp"/>`. A red rectangular box highlights the `Button` element. A red arrow points from a text box below to the `Button` element. The text box contains the instruction: 'Agregar un botón con texto "Botón 1" y establecer sus dimensiones y margen de esta manera'. The XML code is as follows:

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical">

    <Button
        android:layout_width="match_parent"
        android:layout_height="150dp"
        android:text="Botón 1"
        android:layout_margin="10dp"/>

</LinearLayout>
```

Agregar un botón con texto "Botón 1" y establecer sus dimensiones y margen de esta manera

activity_main.xml x MainActivity.java x

LinearLayout

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical">

    <Button
        android:layout_width="match_parent"
        android:layout_height="150dp"
        android:text="Botón 1"
        android:layout_margin="10dp"/>

    <Button
        android:layout_width="match_parent"
        android:layout_height="150dp"
        android:text="Botón 2"
        android:layout_margin="10dp"/>

    <Button
        android:layout_width="match_parent"
        android:layout_height="150dp"
        android:text="Botón 3"
        android:layout_margin="10dp"/>

    <Button
        android:layout_width="match_parent"
        android:layout_height="150dp"
        android:text="Botón 4"
        android:layout_margin="10dp"/>

</LinearLayout>
```

Copiar y pegar
tres veces para
definir así los
botones 2, 3 y 4

Ejecutar en un emulador

```
C <LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical">
    <Button
        android:layout_width="match_parent"
        android:layout_height="150dp"
        android:text="Botón 1"
        android:layout_margin="10dp"/>
    <Button
        android:layout_width="match_parent"
        android:layout_height="150dp"
        android:text="Botón 2"
        android:layout_margin="10dp"/>
    <Button
        android:layout_width="match_parent"
        android:layout_height="150dp"
        android:text="Botón 3"
        android:layout_margin="10dp"/>
    <Button
        android:layout_width="match_parent"
        android:layout_height="150dp"
        android:text="Botón 4"
        android:layout_margin="10dp"/>
</LinearLayout>
```

Ejecutar en un emulador

¿ Se visualizan
adecuadamente todos los
botones ?

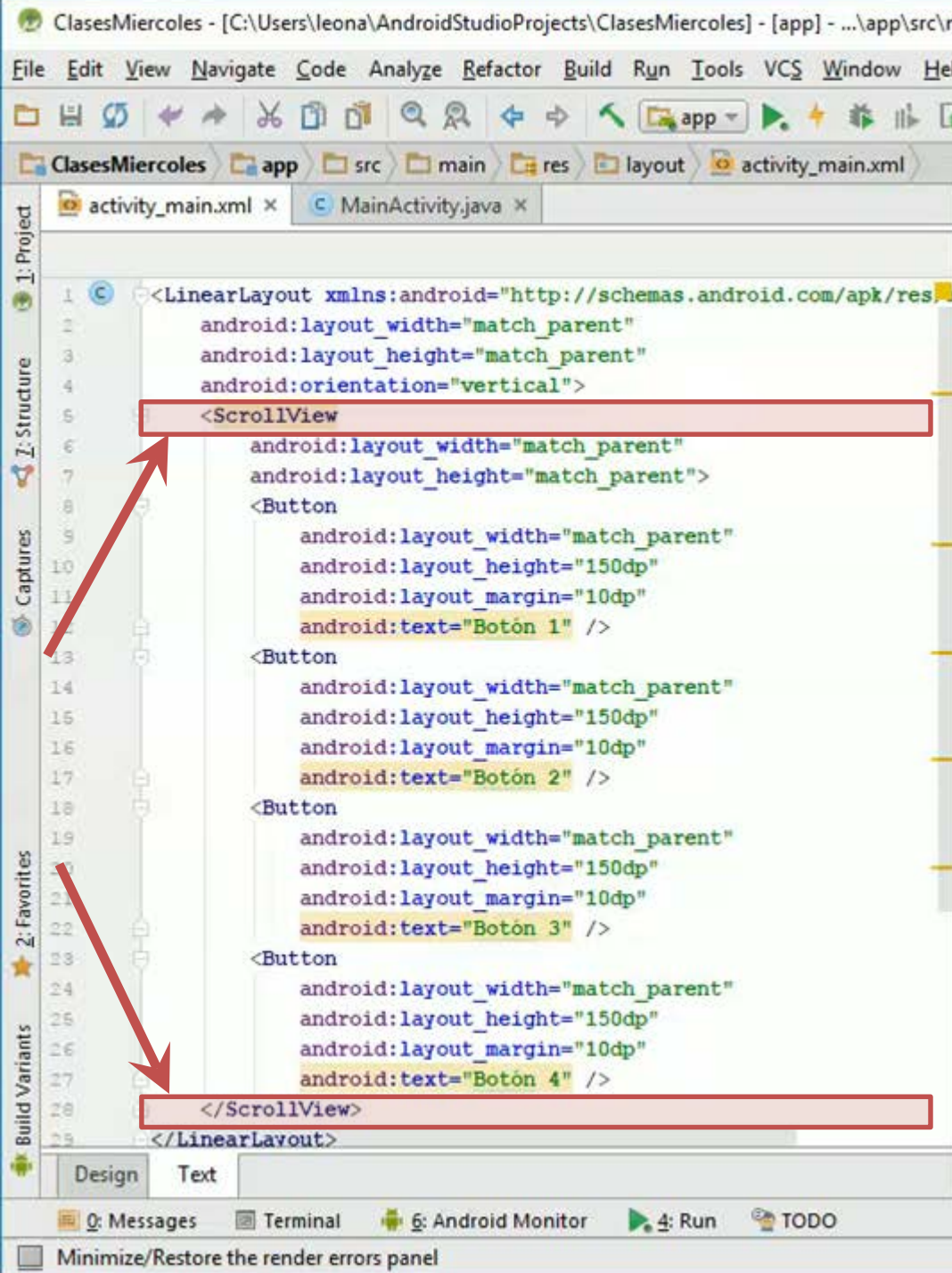
```
<?xml version="1.0" encoding="utf-8" ?>
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical">
    <Button
        android:layout_width="match_parent"
        android:layout_height="150dp"
        android:text="Botón 1"
        android:layout_margin="10dp"/>
    <Button
        android:layout_width="match_parent"
        android:layout_height="150dp"
        android:text="Botón 2"
        android:layout_margin="10dp"/>
    <Button
        android:layout_width="match_parent"
        android:layout_height="150dp"
        android:text="Botón 3"
        android:layout_margin="10dp"/>
    <Button
        android:layout_width="match_parent"
        android:layout_height="150dp"
        android:text="Botón 4"
        android:layout_margin="10dp"/>
</LinearLayout>
```

Ejecutar en un emulador

<ScrollView>

A1 rescate

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical">
    <ScrollView>
        <Button
            android:layout_width="match_parent"
            android:layout_height="150dp"
            android:text="Botón 3"
            android:layout_margin="10dp"/>
        <Button
            android:layout_width="match_parent"
            android:layout_height="150dp"
            android:text="Botón 4"
            android:layout_margin="10dp"/>
    </ScrollView>
</LinearLayout>
```



Pregunta:
¿Por qué esta
solución no podría
ser válida?

Respuesta:
Porque **ScrollView** es
un **FrameLayout**
especializado que sólo
puede alojar un hijo
directo

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    >
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <Button
            android:layout_width="match_parent"
            android:layout_height="150dp"
            android:layout_margin="10dp"
            android:text="Botón 1" />

        <Button
            android:layout_width="match_parent"
            android:layout_height="150dp"
            android:layout_margin="10dp"
            android:text="Botón 2" />

        <Button
            android:layout_width="match_parent"
            android:layout_height="150dp"
            android:layout_margin="10dp"
            android:text="Botón 3" />

        <Button
            android:layout_width="match_parent"
            android:layout_height="150dp"
            android:layout_margin="10dp"
            android:text="Botón 4" />

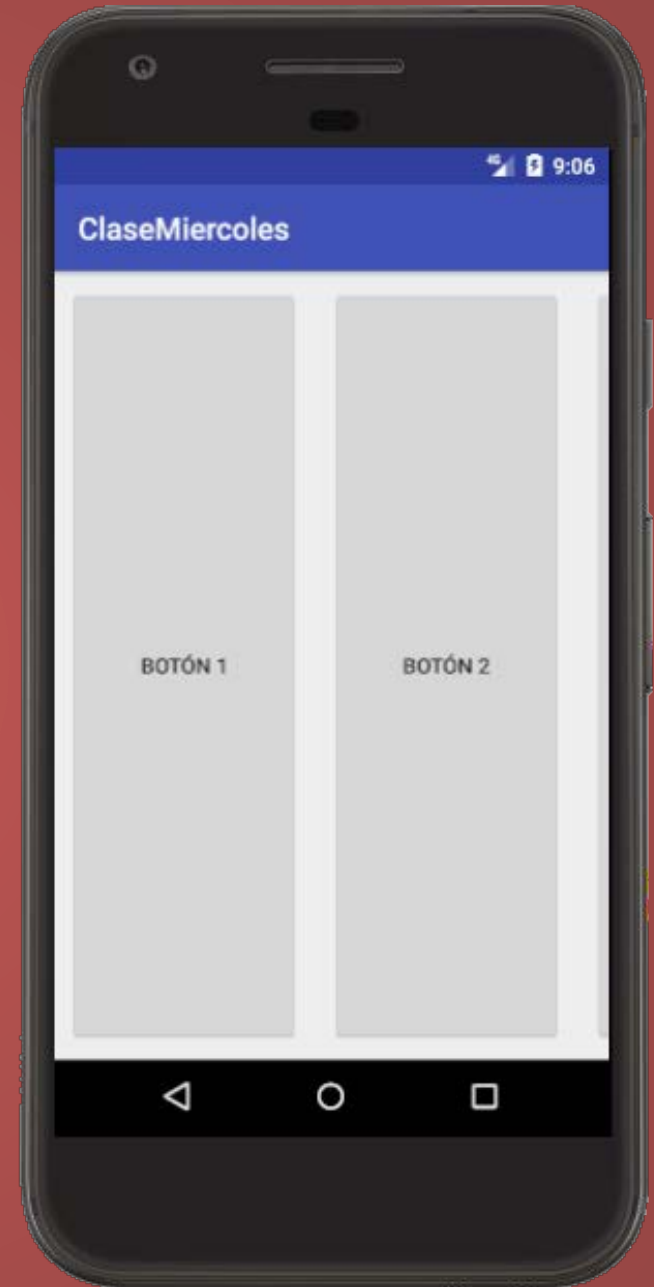
    </LinearLayout>
</ScrollView>
```

Solución válida:
El **ScrollView** aloja
un único hijo directo
(un **LinearLayout**
que contiene a los
botones)

Modificar la aplicación para
disponer los botones
horizontalmente.

El scroll ahora debe ser
horizontal.

El ancho de los botones debe
ser de 150dp



```
<HorizontalScrollView xmlns:android="http://schemas.android.
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="horizontal">
        <Button
            android:layout_width="150dp"
            android:layout_height="match_parent"
            android:layout_margin="10dp"
            android:text="Botón 1" />
        <Button
            android:layout_width="150dp"
            android:layout_height="match_parent"
            android:layout_mar
            android:text="Botó
        <Button
            android:layout_wid
            android:layout_hei
            android:layout_mar
            android:text="Botó
```

Solución:
Se debe utilizar un
<HorizontalScrollView>

Vamos a acceder
programáticamente a los
elementos visuales (vistas o
views) de la activity

Para identificar los **views** del
layout vamos a usar el atributo **id**

Atributo id

.xml

```
<Button
```

```
    android:id="@+id/boton1"  
    android:layout_width="150dp"  
    android:layout_height="match_parent"  
    android:layout_margin="10dp"  
    android:text="Botón 1" />
```

El **id** es un atributo especial que permite identificar al elemento desde el código java

.java

```
@Override
```

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    Button b = (Button) this.findViewById(R.id.boton1);  
    ...  
}
```

```
<HorizontalScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
```

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

```
<Button
```

```
    android:id="@+id/boton1"
```

```
    android:layout_width="150dp"
```

```
    android:layout_height="match_parent"
```

```
    android:layout_margin="10dp"
```

```
    android:text="Botón 1" />
```

```
<Button
```

```
    android:id="@+id/boton2"
```

```
    android:layout_width="150dp"
```

```
    android:layout_height="match_parent"
```

```
    android:layout_margin="10dp"
```

```
    android:text="Botón 2" />
```

```
<Button
```

```
    android:id="@+id/boton3"
```

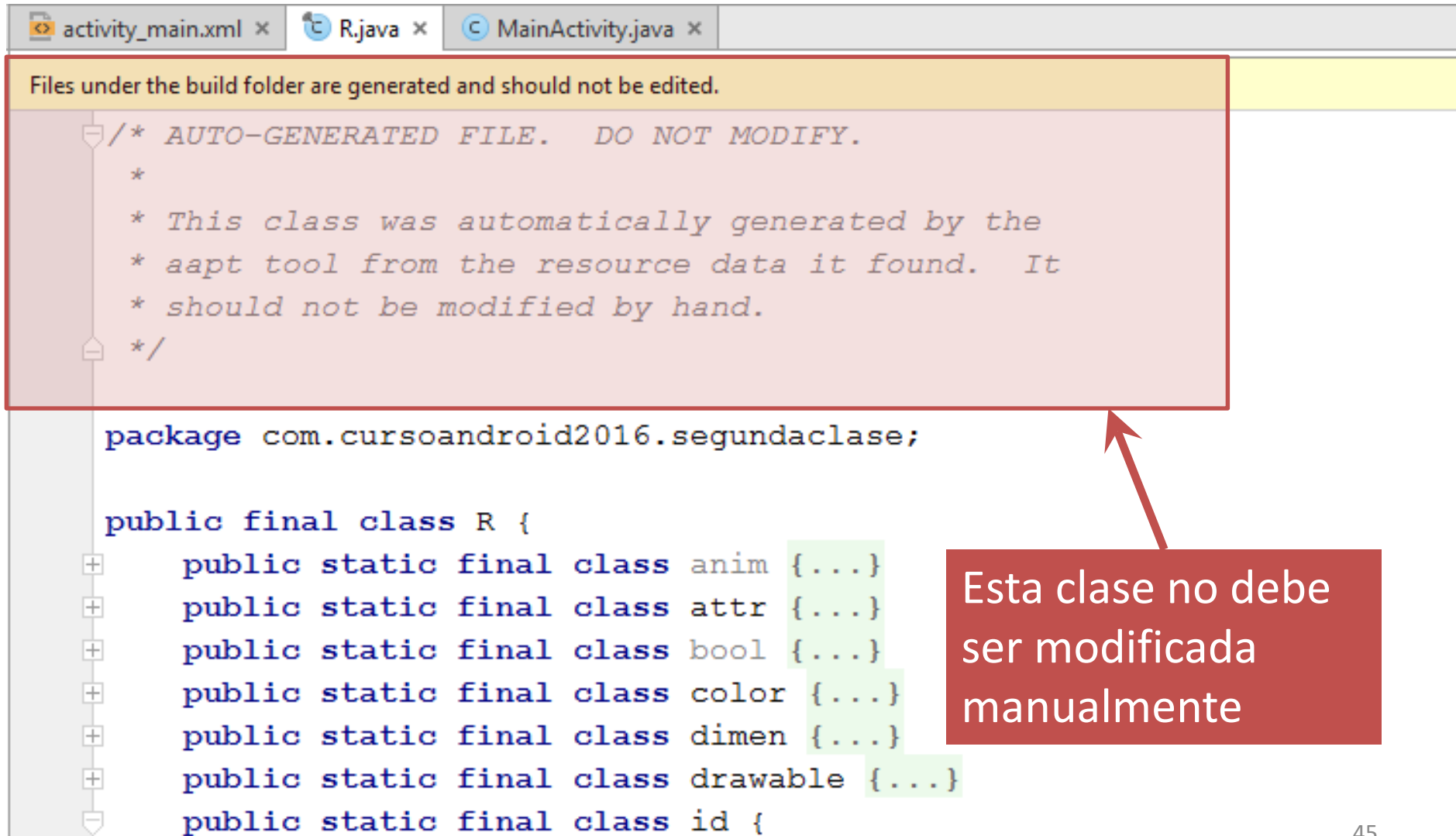
```
    android:layout_width="150dp"
```

Agregar el atributo
android:id a
cada uno de los
botones

¿ Qué significa "@+id/boton1" ?

- Al tipear **@+id/** el entorno convierte una etiqueta en un recurso con un nombre determinado.
- Así al tipear **"@+id/boton1"** se crea un recurso llamado **boton1** que luego puede referenciarse desde el código java por medio de la clase estática **R.id**
- La clase estática **R** y sus clases miembros anidadas se generan automáticamente
- Luego de definir los **ids** para los cuatro botones podemos encontrarlos en el archivo **R.java** autogenerado.

Archivo R.java



activity_main.xml x R.java x MainActivity.java x

Files under the build folder are generated and should not be edited.

```
/* 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 com.cursoandroid2016.segundaclass;

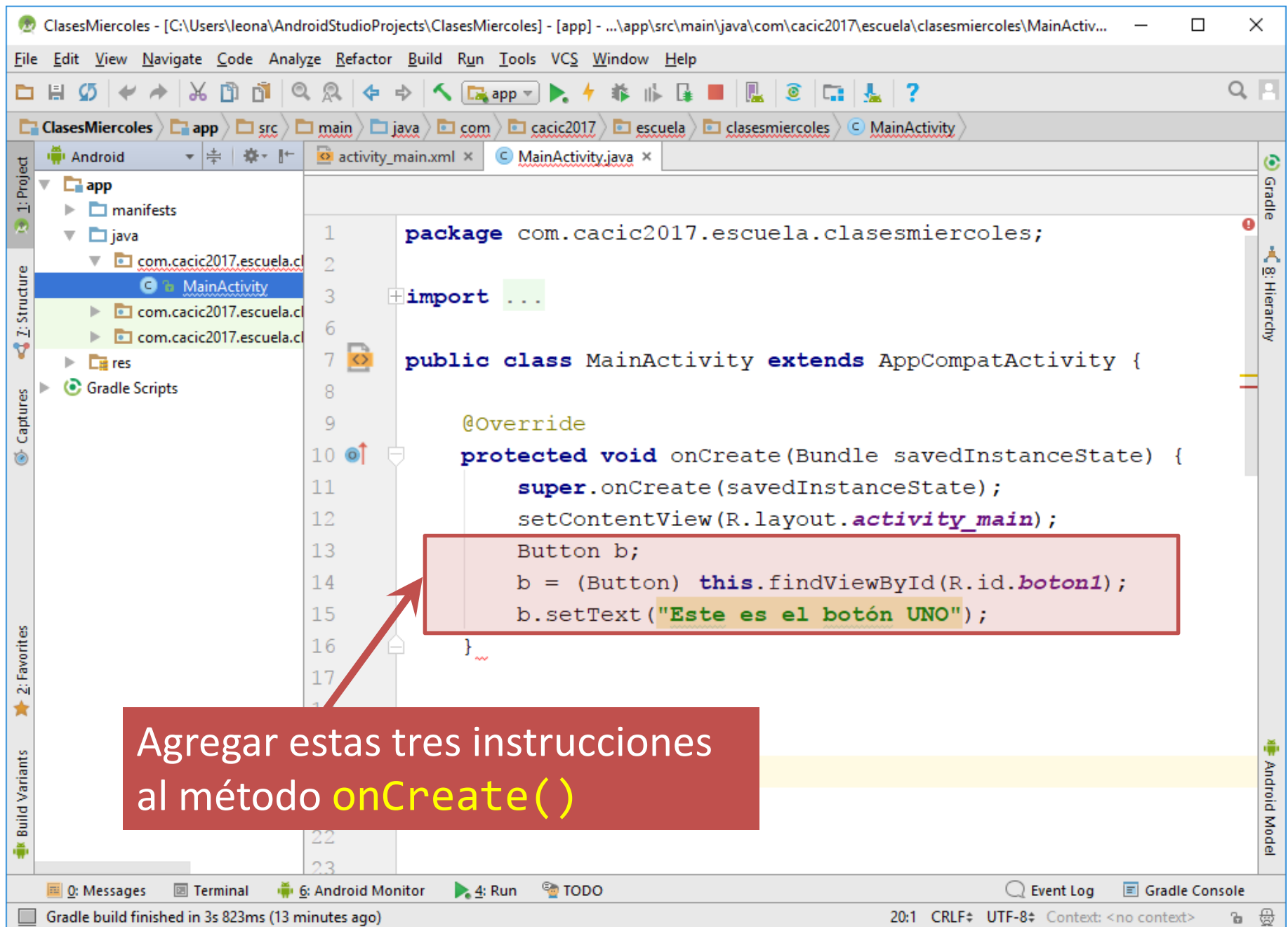
public final class R {
    public static final class anim {...}
    public static final class attr {...}
    public static final class bool {...}
    public static final class color {...}
    public static final class dimen {...}
    public static final class drawable {...}
    public static final class id {
```

Esta clase no debe ser modificada manualmente

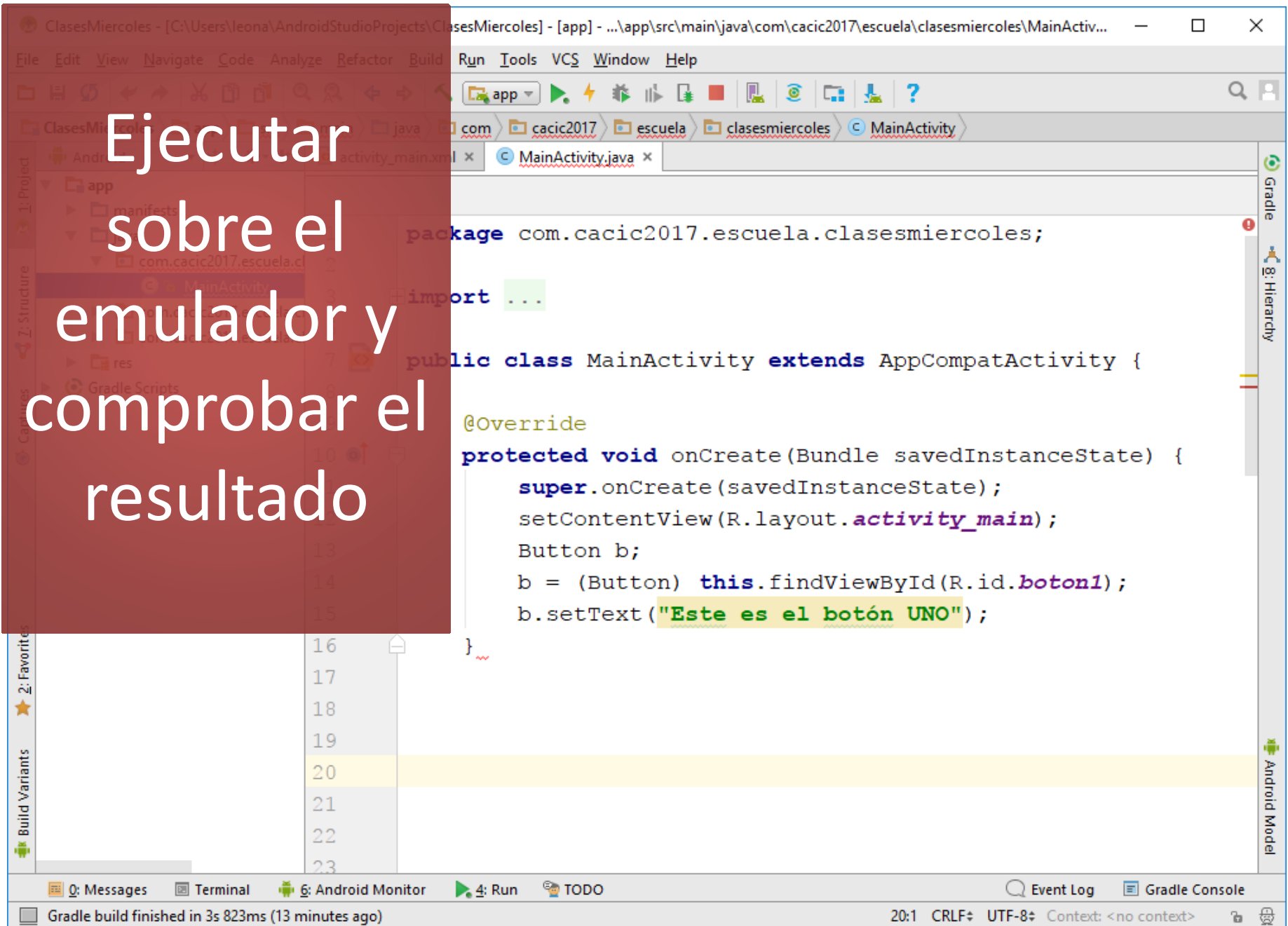
Miembros de la clase R.id

build folder are generated and should not be edited.

```
public static final int add=0x7f0b0013;
public static final int alertTitle=0x7f0b0033;
public static final int always=0x7f0b001d;
public static final int beginning=0x7f0b001a;
public static final int boton1=0x7f0b0054;
public static final int boton2=0x7f0b0055;
public static final int boton3=0x7f0b0056;
public static final int boton4=0x7f0b0057;
public static final int bottom=0x7f0b0022;
public static final int buttonPanel=0x7f0b002e;
public static final int cancel_action=0x7f0b0059;
public static final int checkbox=0x7f0b003c;
public static final int chronometer=0x7f0b005f;
public static final int collapseActionView=0x7f0b001e;
public static final int contentPanel=0x7f0b0034;
public static final int custom=0x7f0b003a;
public static final int customPanel=0x7f0b0039;
```



Ejecutar
sobre el
emulador y
comprobar el
resultado



Ejecutar
sobre el
emulador y
comprobar el
resultado



Explicación del código Java

Se define la variable **b** de tipo **Button**

Se asigna a la variable **b** el objeto **View** de la **activity** cuyo **id** es **boton1** (es necesario *castear* a **Button** porque **findViewById** devuelve un objeto **View**)

```
Button b;
```

```
b = (Button) this.findViewById(R.id.boton1);
```

```
b.setText("Este es el botón UNO");
```

Se establece el texto del **botón 1**