

SEMINARIO DE LENGUAJES

OPCIÓN ANDROID



Intents.

Esp. Fernández Sosa Juan Francisco

Activación de Componentes

- Las actividades de una aplicación son capaces de iniciar componentes de la misma aplicación como también de otras aplicaciones que estén disponibles en el dispositivo.
- Sin embargo, esto **no** es posible hacerlo de **forma directa** ya que cada aplicación se ejecuta en un proceso independiente con permisos de archivos que limitan el acceso a otras aplicaciones.
- Para hacerlo, la actividad debe **pedírselo al S.O.** por medio de un mensaje especificando la intención (***intent***) de iniciar un componente específico. Luego, es el propio Android el que activa ese componente.

Ejemplo práctico

- Vamos a crear un proyecto con dos *activities*.
- Al presionar un botón en la *activity* principal se mostrará información de la aplicación en una *activity* distinta.

Ejemplo práctico

- Crear un nuevo proyecto basado en un **Empty Activity** y definir el siguiente **layout**:

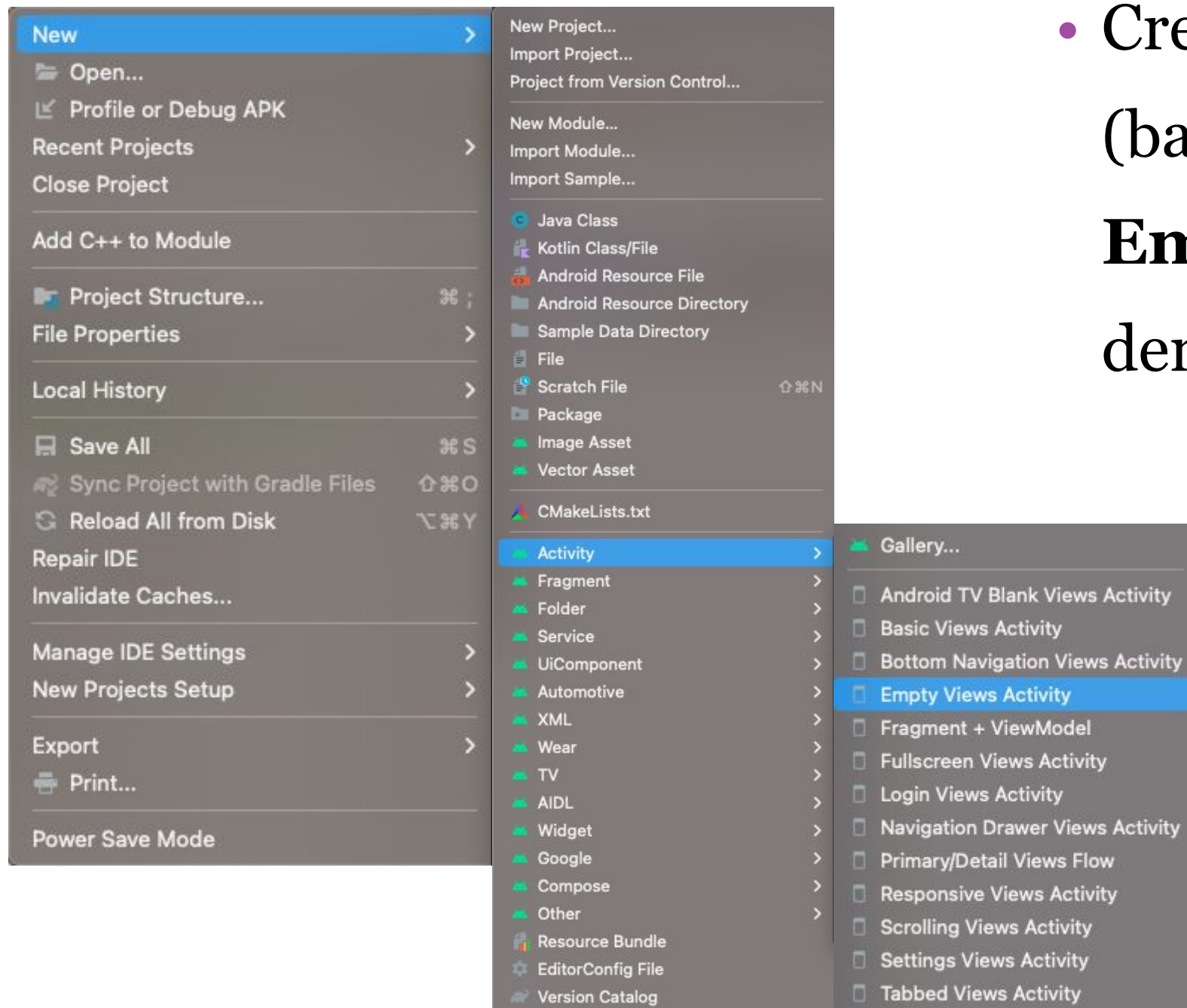
```
<?xml version="1.0" encoding="utf-8" ?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Mostrar Información"
        android:onClick="mostrarInfo"
    />

</LinearLayout>
```

En **onClick** se especifica el nombre del método de la **activity** que se ejecutará al presionar el botón

Ejemplo práctico



- Crear una segunda *activity* (basada en la plantilla **Empty Views Activity**) y denominarla **InfoActivity**

Ejemplo práctico

- Definir este layout para **InfoActivity**

The screenshot shows the Android Studio IDE with the `activity_info.xml` file open. The XML code defines a `LinearLayout` containing a `TextView`. The `TextView` has a large font size and a specific ID. A callout box explains the hex color code used for the background. Another callout box highlights the ID definition for the `TextView`.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#70FF90"
    xmlns:android="http://schemas.android.com/apk/res/android">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="30sp"
        android:text=" Info del Curso de Android "
        android:id="@+id/texto"
    />

</LinearLayout>
```

Callout 1: Hex color code breakdown

Hex	R	G	B
#70FF90	70	FF	90

Callout 2: Important: definir un id para el **TextView**

Ejemplo práctico

- Ya tenemos definida la *activity* que queremos mostrar al presionar el botón "**Mostrar Información**" en la *activity* principal
- Ahora debemos programar el manejador del evento **onClick** de dicho botón

Agregar el método `mostrarInfo` en la clase `MainActivity`

Este método lo establecimos en el `onClick` del botón de esta *activity*
(análisis de este código en la siguiente diapositiva)

```
import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
    fun mostrarInfo(v: View) {
        val i = Intent(this, InfoActivity::class.java)
        startActivity(i)
    }
}
```


Se crea el `intent i` que referencia a la *activity* que se va a iniciar



```
fun mostrarInfo(v: View) {  
    val i = Intent(this, InfoActivity::class.java);  
    startActivity(i)  
}
```



Se inicia la nueva *activity* por medio del `Intent i`

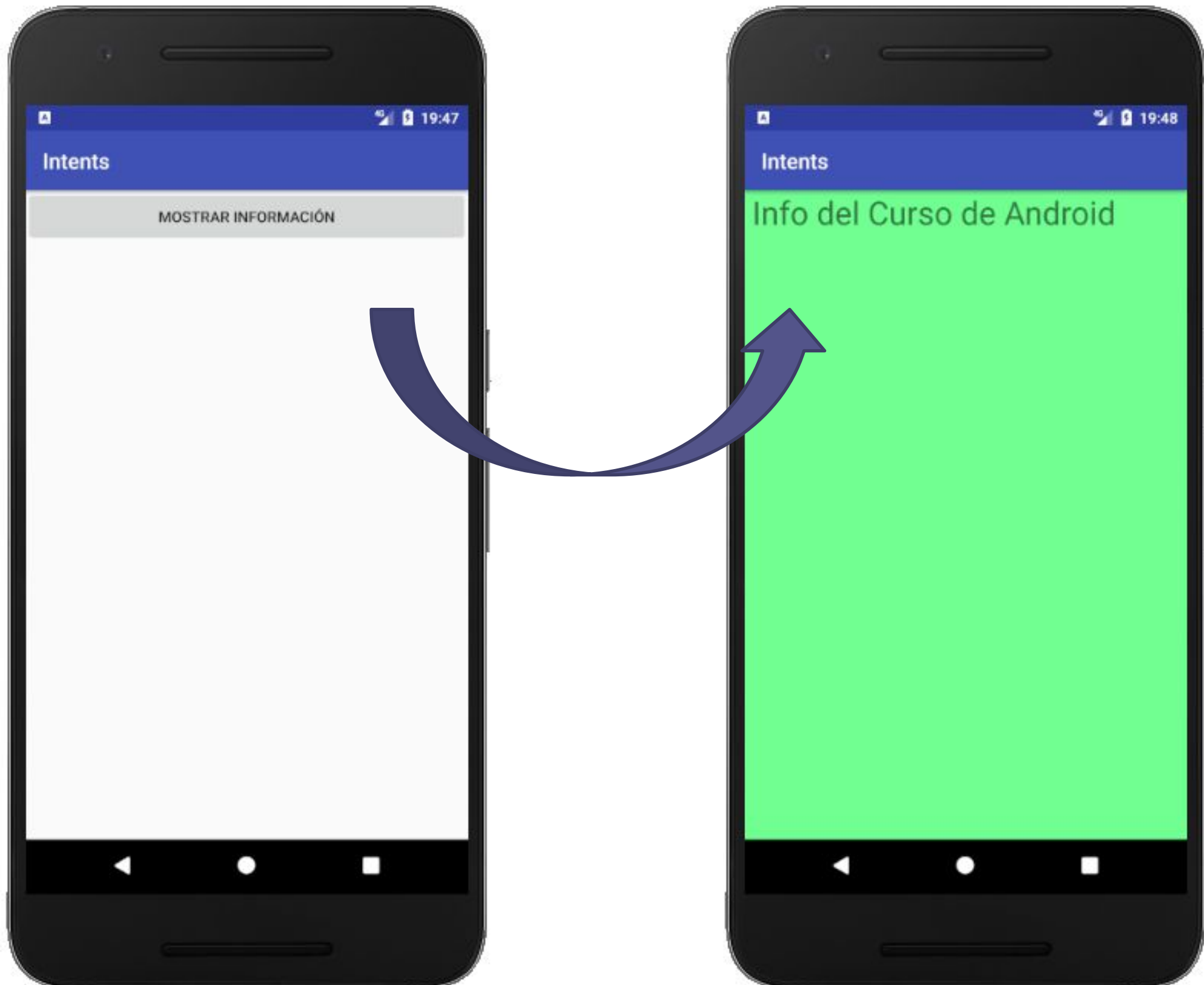
Contexto desde el que se va a iniciar **InfoActivity**, en este caso **this** hace referencia a la *activity* principal (instancia de **MainActivity**)

```
fun mostrarInfo(v: View) {  
    val i = Intent(this, InfoActivity::class.java)  
    startActivity(i)  
}
```

Referencia a la clase de la *activity* que se va a iniciar

Ejecutar en el emulador para
comprobar comportamiento

Resultado en el emulador



Pasando información a la *activity* iniciada

- Para pasar información a la *activity* se utiliza el mismo **Intent** con el que se la inicia.
- Se pueden pasar tantos datos como se requieran llamando repetidamente al método **putExtra()** del **Intent**
- **putExtra()** recibe dos parámetros: un **String** (a modo de clave) y el dato en cuestión.
- Los datos enviados podrán ser recuperados en la *activity* que se inicia por medio de una referencia a la clase **intent**

Modificar el método mostrarInfo de MainActivity

```
fun mostrarInformacion(v: View) {  
    val i = Intent(this, InfoActivity::class.java);  
    i.putExtra("dato1", "La cantidad de alumnos es: ");  
    i.putExtra("dato2", 85);  
    startActivity(i)  
}  
}
```

Agregar estas líneas para enviar a la *activity* que se inicia dos datos, un String y un entero




Agregar en el método onCreate de InfoActivity

(análisis de este código en la siguiente diapositiva)

```
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.TextView
```

```
class InfoActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_info)
        val extras = intent.extras;
        val mensaje = extras?.getString("dato1");
        val cantidad = extras?.getInt("dato2");
        val texto= findViewById<TextView>(R.id.texto);
        texto.setText(mensaje+cantidad)
    }
}
```



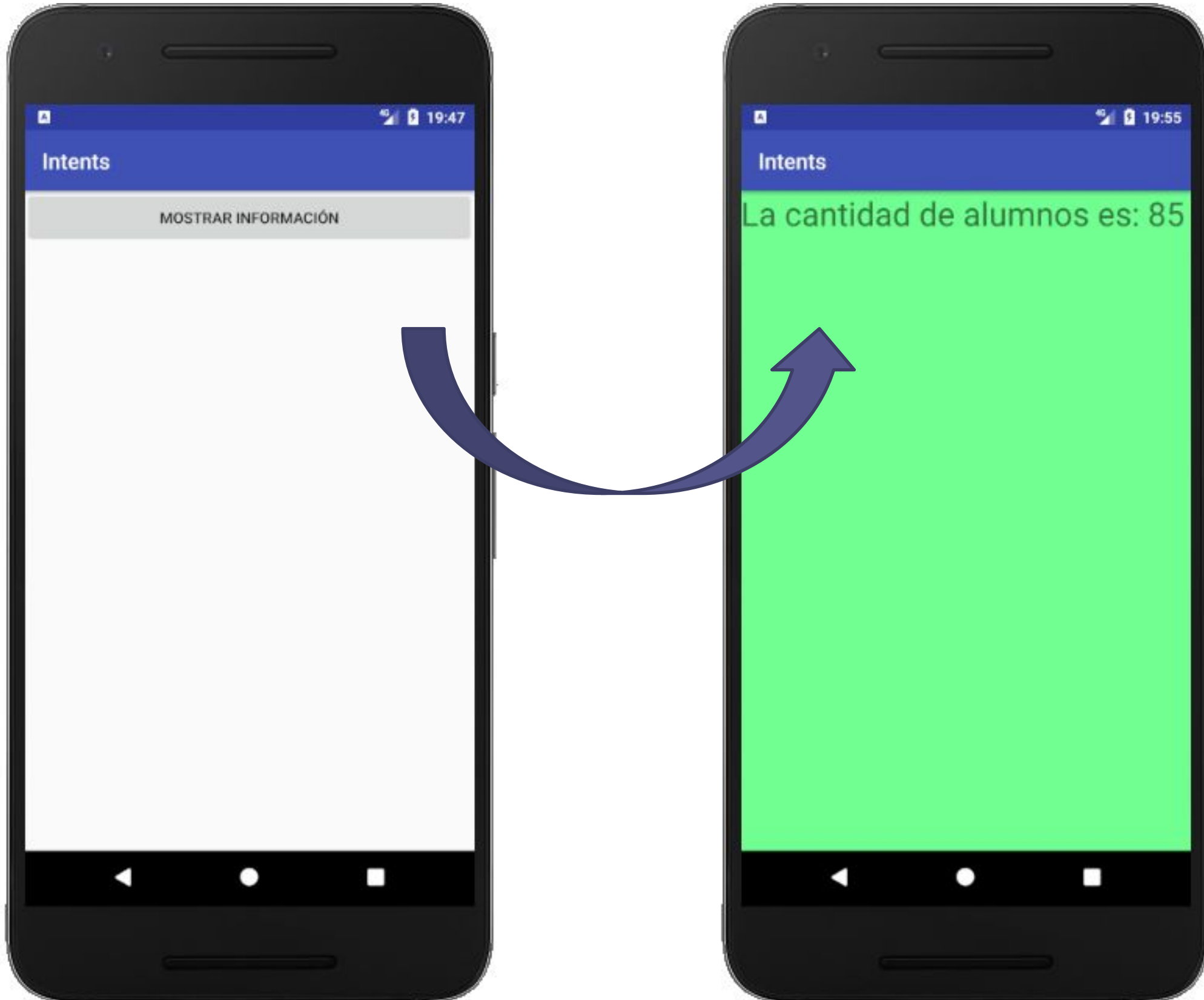
Recupero los extras que se adjuntaron en el **intent** y lo almaceno en una constante, ya que no se modificará su valor..
A dicha constante puedo pedirle el **valor** de los parámetros a partir de la **clave** definida al momento de enviarlos

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_main2)  
    val extras = intent.extras;  
    val mensaje = extras?.getString("dato1");  
    val cantidad = extras?.getInt("dato2");  
    val texto= findViewById<TextView>(R.id.texto);  
    texto.setText(mensaje+cantidad)  
}
```

Obtenemos la referencia al **TextView**, a partir de su ID, y establecemos el texto a mostrar

Ejecutar en el
emulador

Resultado en el emulador

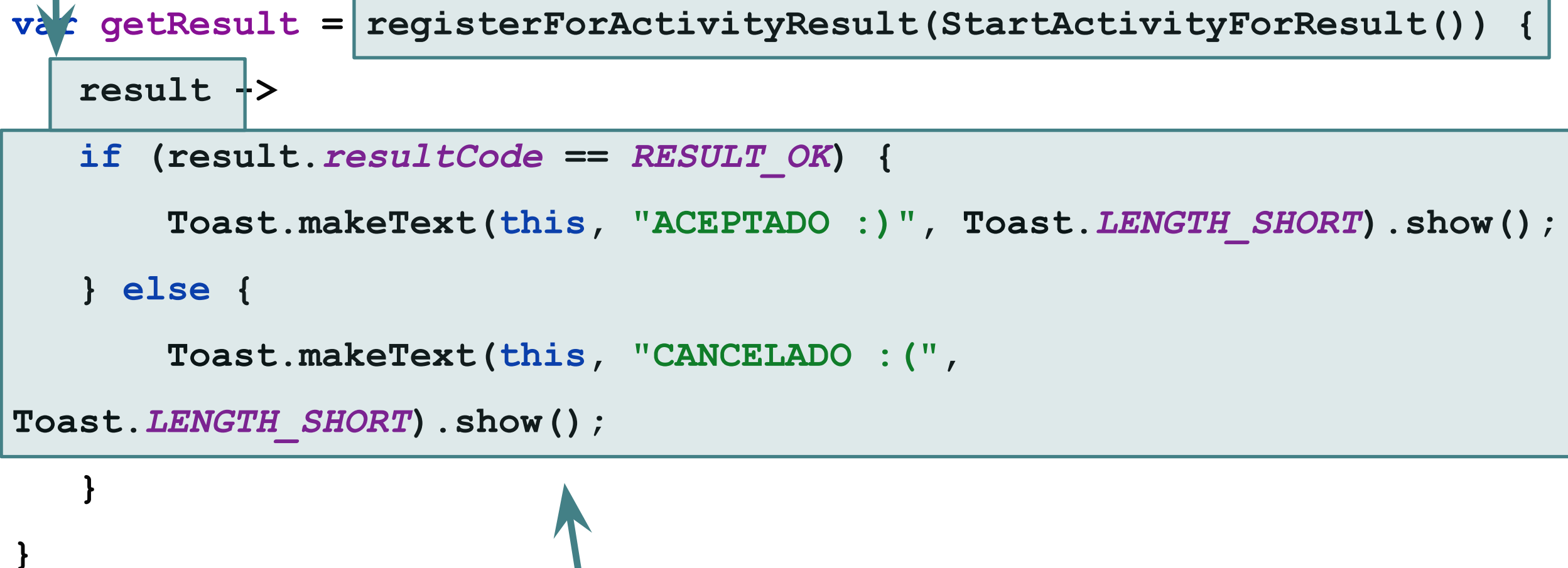


Devolviendo información a la *activity* iniciadora

- Para recibir un resultado desde la *activity* que se va a iniciar se debe lanzar dicha *activity* utilizando un objeto de la clase “**ActivityResultLauncher**”.
- Este objeto permite lanzar la actividad y definir o registrar un escuchador de evento, que será llamado, por el sistema operativo, cuando la actividad lanzada finalice. Este objeto puede definirse como un atributo de la clase.

Parámetro que contiene los posibles datos devueltos por la actividad, además del código de respuesta.

Declaración de un objeto de la clase
“ActivityResultLauncher”



```
var getResult = registerForActivityResult(StartActivityForResult()) {  
    result ->  
    if (result.resultCode == RESULT_OK) {  
        Toast.makeText(this, "ACEPTADO :", Toast.LENGTH_SHORT).show();  
    } else {  
        Toast.makeText(this, "CANCELADO :(",  
Toast.LENGTH_SHORT).show();  
    }  
}
```

Bloque de código que se ejecutará al retornar de la actividad

```
fun mostrarInformacion(v: View) {  
    val i = Intent(this, InfoActivity::class.java);  
    getResult.launch(i);  
}
```

Para lanzar la actividad, se debe reemplazar el método “**startActivity()**” por el método “**launch()**” del objeto **ActivityResultLauncher**, enviando como parámetro el **Intent**.

Valores posibles de resultCode

- El atributo **resultCode** obtenido en como resultado del método el método **registerForActivityResult()** es un entero, por lo tanto el programador puede establecer arbitrariamente tanto su valor como su interpretación.
- Sin embargo se ha convenido lo siguiente :
0 = Cancelado -1 = Aceptado.
- En **Activity.class** se encuentran definidas las siguientes constantes:

```
/** Standard activity result: operation canceled. */  
public static final int RESULT_CANCELED      = 0;  
/** Standard activity result: operation succeeded. */  
public static final int RESULT_OK            = -1;
```

Devolviendo información a la *activity* iniciadora

- En **Activity_info.xml** establecer la orientación del **LinearLayout** en **vertical** y agregar dos botones, un botón "**Aceptar**" y otro "**Cancelar**"

```
<Button
```

```
    android:textSize="30sp"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="Aceptar"  
    android:onClick="aceptarYcerrar"  
/>
```

```
<Button
```

```
    android:textSize="30sp"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="Cancelar"  
    android:onClick="cancelarYcerrar"  
/>
```

Devolviendo información a la *activity* iniciadora

- En `InfoActivity.kt` codificar los siguientes métodos

```
fun aceptarYcerrar(v: View) {  
    setResult(RESULT_OK) ;  
    this.finish() ;  
}
```

```
fun cancelarYcerrar(v: View) {  
    setResult(RESULT_CANCELED) ;  
    this.finish() ;  
}
```

Establece resultado



Cierra la *activity*

Correr en el
emulador

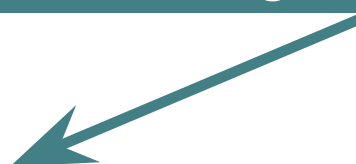
Devolviendo información a la *activity* iniciadora

- Vamos a modificar la aplicación que estamos construyendo agregando un **EditText** en **InfoActivity** para que el usuario pueda ingresar un texto en él.
- Sólo si el usuario presiona el botón "Aceptar", una vez cerrada **InfoActivity**, **MainActivity** debe mostrar en un mensaje **Toast** el texto tipeado por el usuario.

Devolviendo información a la *activity* iniciadora

- Agregar el **EditText** en **Activity_info.xml**

Es necesario establecer un **id** para luego poder referenciar desde el código Kotlin



```
<EditText  
    android:id="@+id/editor"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:hint="Ingresa un texto"/>
```

Devolviendo información a la *activity* iniciadora

- En **InfoActivity.kt** modificar el método

```
fun aceptarYcerrar(v: View) {  
    val editor= findViewById<EditText>(R.id.editor) ;  
    val texto = editor.text.toString() ;  
    val intent = Intent() ;  
    intent.putExtra("input", texto) ;  
    setResult(RESULT_OK, intent) ;  
    this.finish() ;  
}
```

Obtiene referencia al editor

Obtiene el texto ingresado

Crea un intent con la información a devolver

Establece resultado y el intent con la información

Devolviendo información a la *activity* iniciadora

- En **MainActivity.kt** modificar el método

```
var getResult = registerForActivityResult(StartActivityForResult()) {  
    result ->  
    if (result.resultCode == RESULT_OK) {  
        val leyenda = result.data?.getStringExtra("input")  
        Toast.makeText(this, leyenda, Toast.LENGTH_SHORT).show();  
    }  
}
```

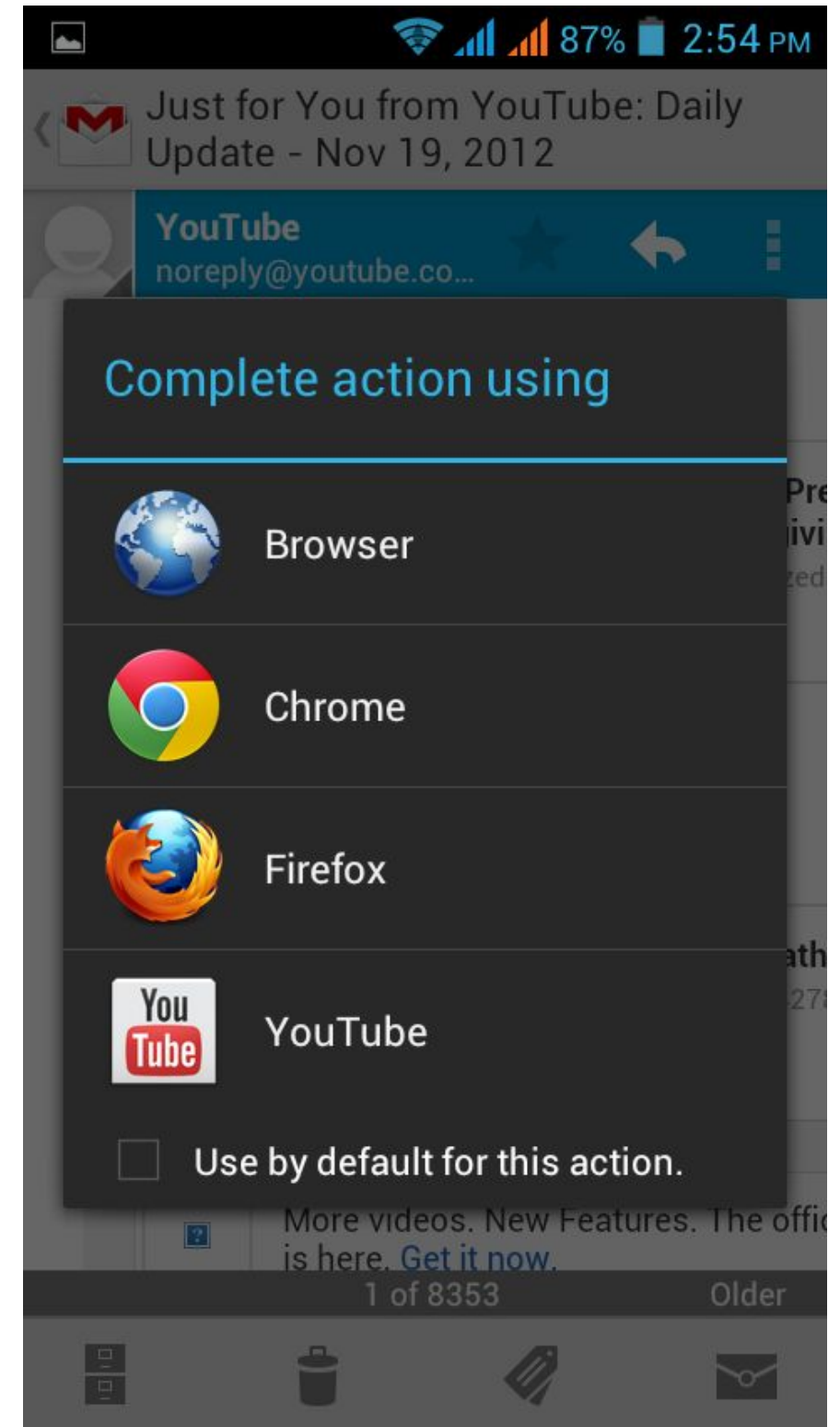
Correr en el
emulador

Intent para llamar *activities* de otras aplicaciones

- Hemos utilizado un objeto **Intent** para iniciar una *activity* de nuestra aplicación de manera **explícita** (**InfoActivity**)
- También existen los **Intent implícitos** que NO especifican el componente que se desea iniciar sino que sólo describen una **acción** a realizar y opcionalmente los datos sobre los que trabajar

Intent para llamar *activities* de otras aplicaciones

- Los **Intent implícitos** delegan en Android la búsqueda de un componente en el dispositivo que pueda realizar la acción especificada. Si hay más de un componente en esas condiciones se presenta un diálogo al usuario para que elija cuál de todos iniciar.



Intent para llamar *activities* de otras aplicaciones

- Android identifica los componentes que pueden responder a un **intent** comparando el contenido del **intent** con los *filtros de intents* que se proporcionan en el archivo de manifiesto de otras aplicaciones instaladas en el sistema

Contenido de los *intents* utilizado para comparar con los filtros de *intents*

- **Acción:** un string que especifica una acción genérica que se debe realizar. Dependiendo de la acción es posible determinar si son necesarios datos extras.
- **Datos:** El objeto **Uri** que hace referencia a los datos en que se debe realizar la acción o el tipo **MIME** de esos datos.
- **Categoría:** un string que contiene información adicional sobre el tipo de componente que debe iniciarse. Un **intent** puede tener varias categorías pero la mayoría de ellos no requieren ninguna
- **Extras:** Pares clave-valor que contienen información adicional necesaria para completar la acción solicitada.

Ejemplo de código que utiliza un Intent implícito para iniciar una *activity* de otra aplicación

```
val i2 = Intent();
```

Se establece la acción del Intent

```
i2.action = Intent.ACTION_SEND;
```

```
i2.type= "text/plain";
```

Se establece el tipo MIME de los datos que se enviarán

```
i2.putExtra(Intent.EXTRA_TEXT, "Texto a enviar");
```

Se agrega al Intent el texto que se enviará a la *activity*

```
if (i2.resolveActivity(packageManager) != null) {
```

```
    startActivity(i2);
```

```
}
```

Se verifica que exista alguna *activity* en el sistema que pueda resolver el intent antes de iniciarla, de lo contrario `startActivity()` fallará

Ejemplo de código que utiliza un **Intent** implícito para iniciar una *activity* de otra aplicación

```
val i2 = Intent();  
i2.action = Intent.ACTION_SEND;  
i2.type= "text/plain";  
i2.putExtra(Intent.EXTRA_TEXT, "Texto a enviar");  
if (i2.resolveActivity(packageManager) != null) {  
    startActivity(i2);  
}
```

Es una constante de tipo **string**
"android.intent.action.SEND"

En la clase **Intent** existen muchas otras
constantes de acciones definidas

Clase que devuelve información
relacionada con los paquetes de las
aplicaciones instaladas en el dispositivo

Es una constante de tipo **string**
"android.intent.extra.TEXT"

La clase **Intent** especifica muchas constantes
EXTRA_* para tipos de datos estandarizados

Intents comunes

```
val i2 = Intent();  
i2.action = Intent.ACTION_SEND;  
i2.type = "text/plain";  
i2.putExtra(Intent.EXTRA_TEXT, "Texto a enviar");  
if (i2.resolveActivity(packageManager) != null) {  
    startActivity(i2);  
}
```

La acción SEND también se la conoce como compartir. Dependiendo de las aplicaciones que se encuentren instaladas en el dispositivo, se mostrará al usuario un cuadro de diálogo con una lista de posibles elecciones



Share with Messages

Just on

Use a different app



Drive
Save to Drive



Nearby Share



Bluetooth



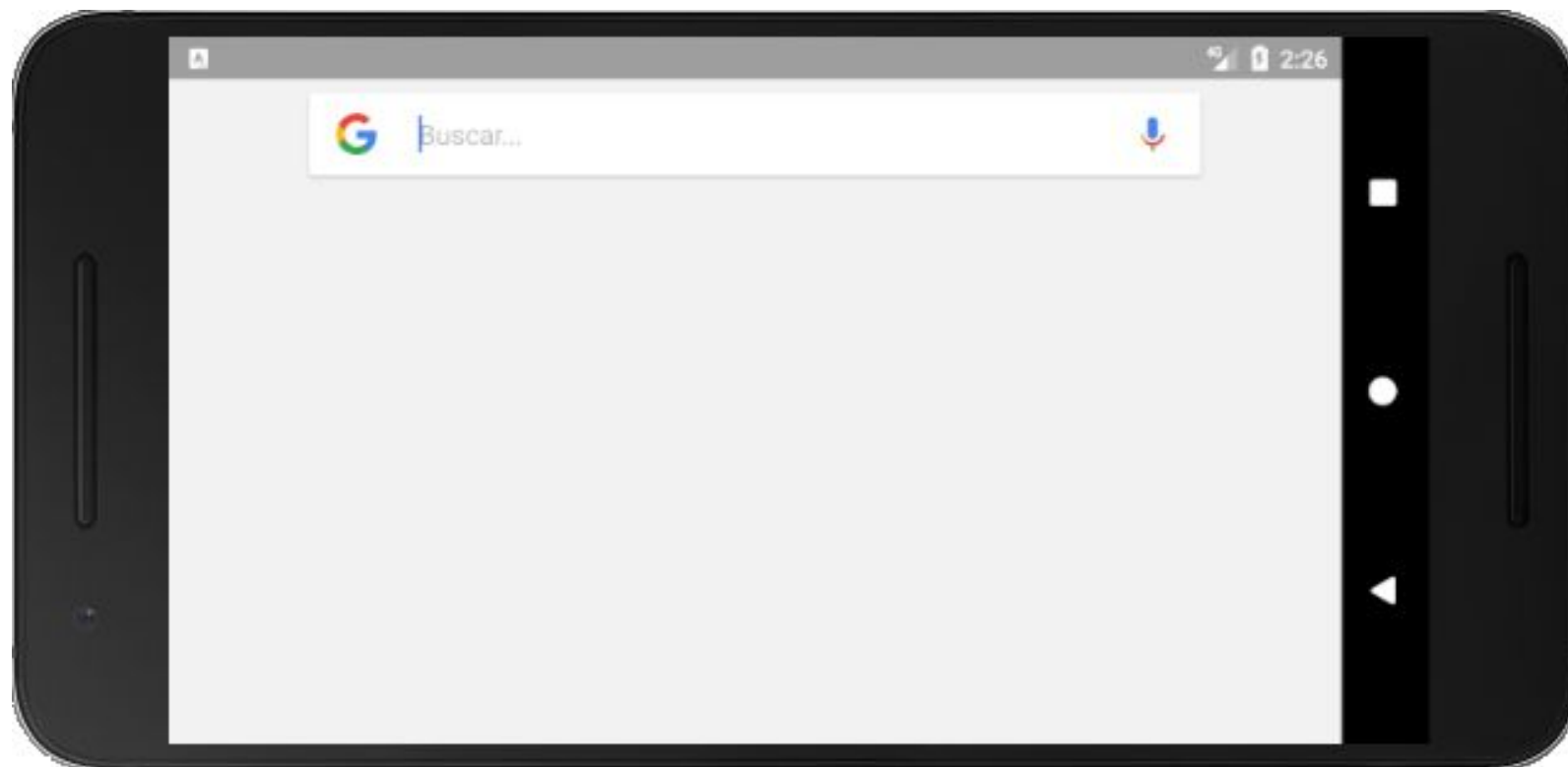
Gmail

Intents comunes

- Existe una gran cantidad de *intents* implícitos que pueden usarse para realizar acciones comunes.
- Modifique la aplicación para implementar los ejemplos que se muestran a partir de la próxima diapositiva.

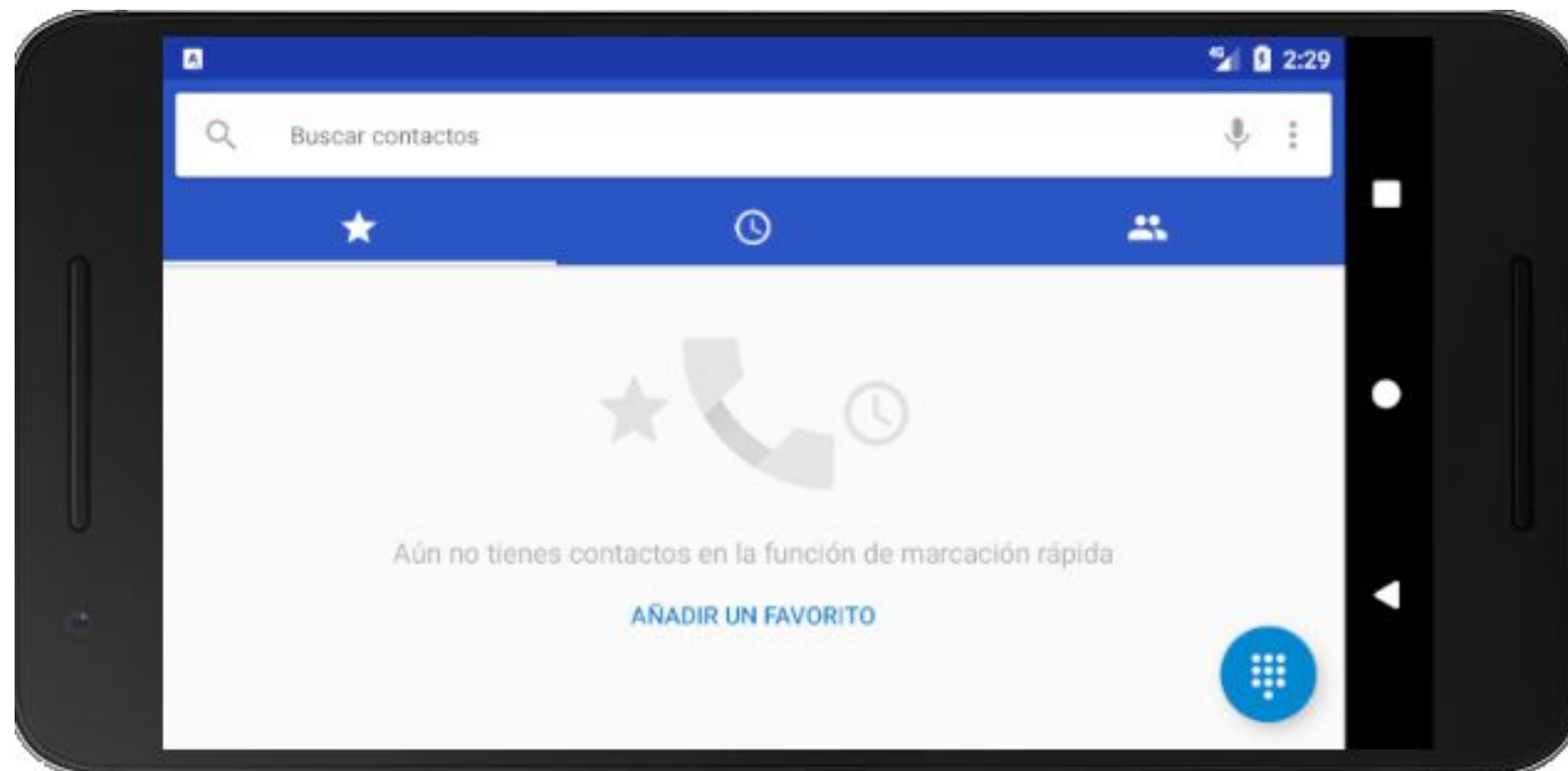
Intents comunes

```
val i2 = Intent();  
i2.action = Intent.ACTION_WEB_SEARCH;  
startActivity(i2);
```



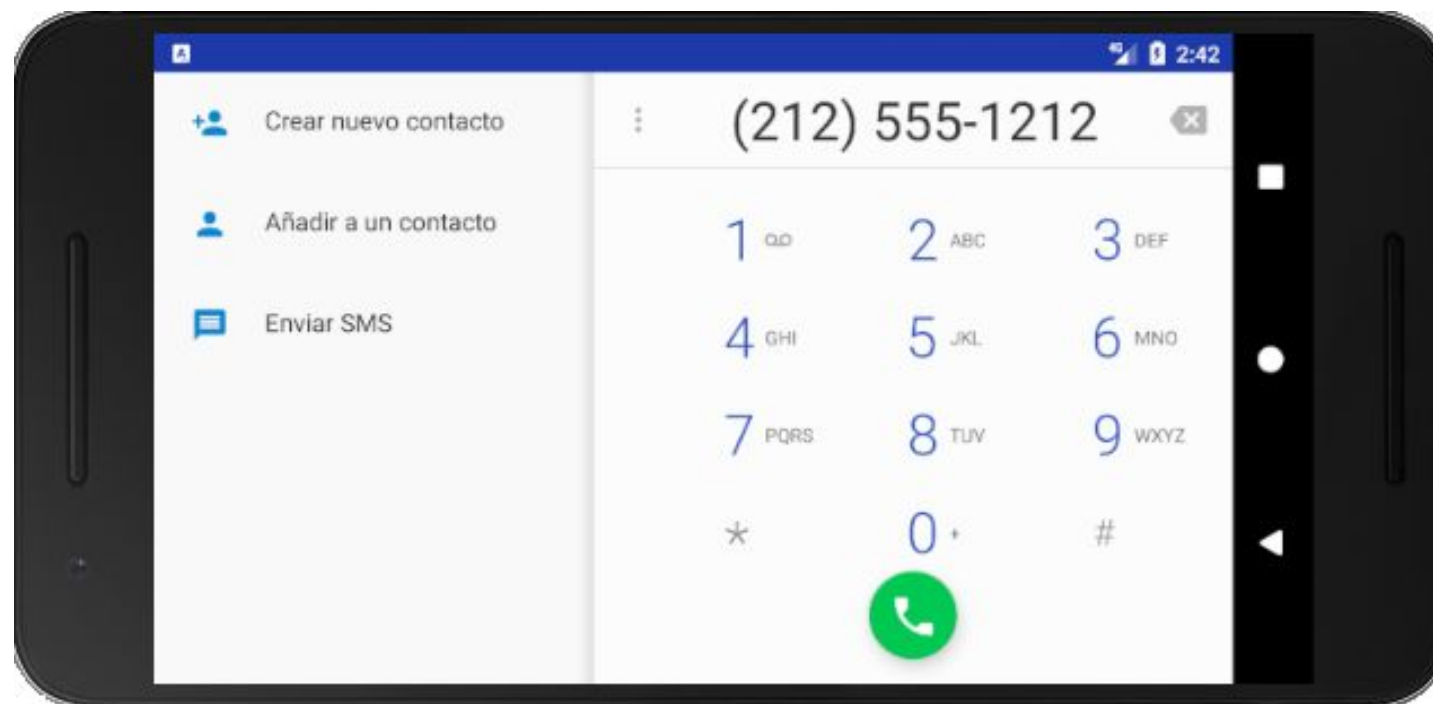
Intents comunes

```
val i2 = Intent();  
i2.action = Intent.ACTION_DIAL;  
startActivity(i2);
```



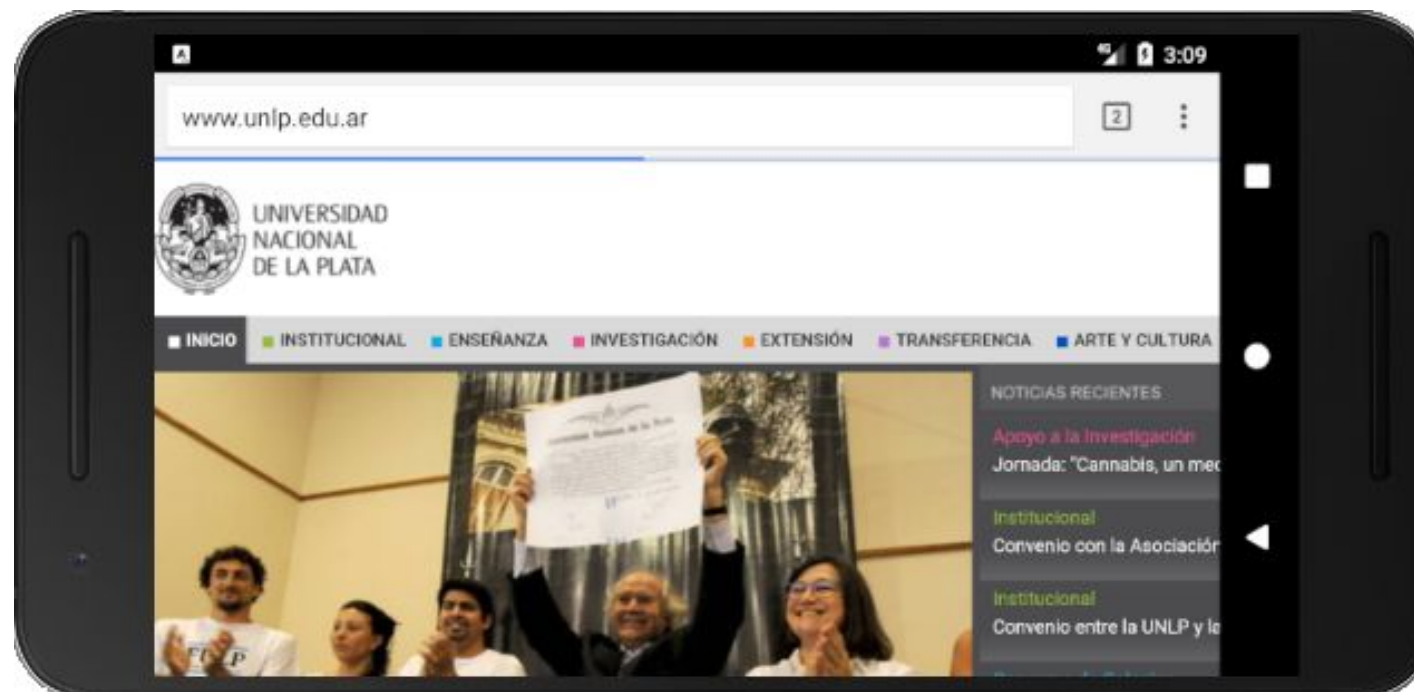
Intents comunes

```
val i2 = Intent();  
i2.action = Intent.ACTION_DIAL;  
i2.data = (Uri.parse("tel:"+"2210303456"));  
startActivity(i2);
```



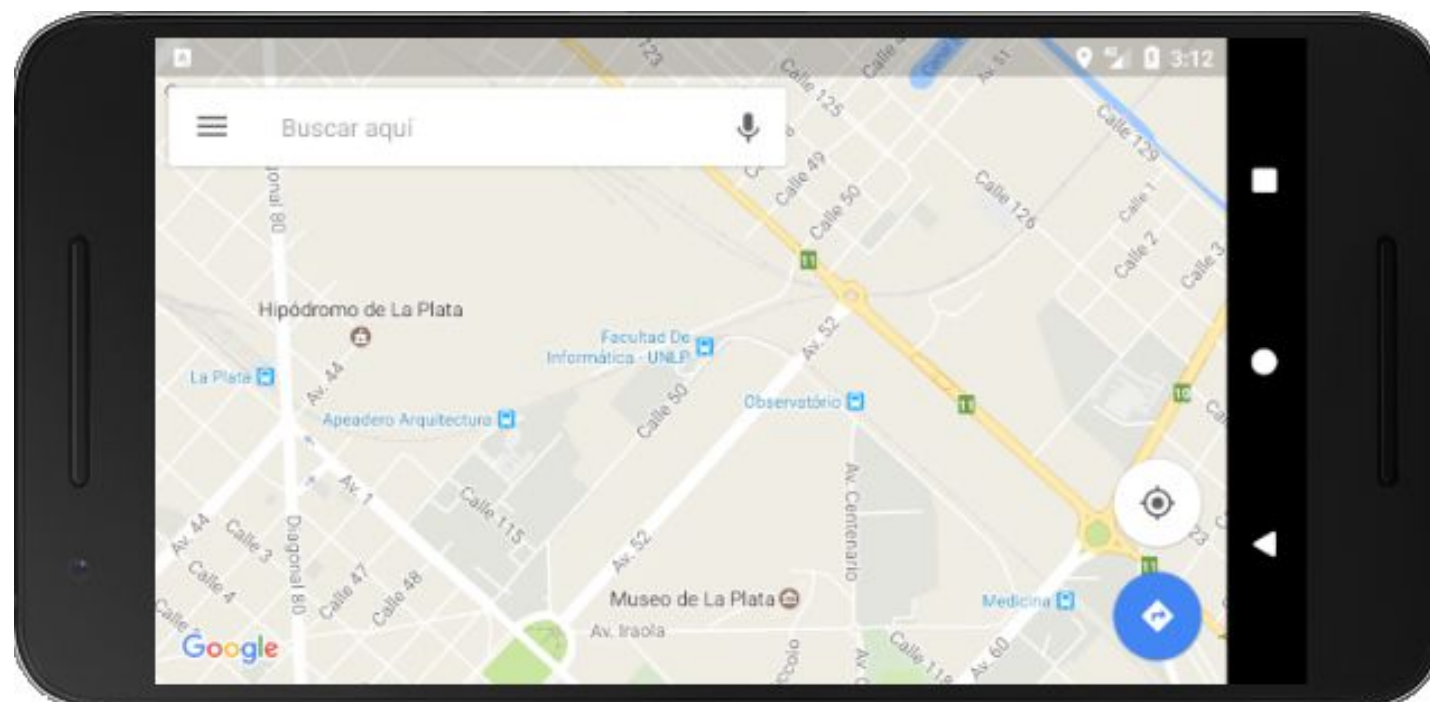
Intents comunes

```
val i2 = Intent();  
i2.action = Intent.ACTION_VIEW;  
i2.data = Uri.parse("https://info.unlp.edu.ar");  
startActivity(i2);
```



Intents comunes

```
val i2 = Intent();  
i2.action = Intent.ACTION_VIEW;  
i2.data = Uri.parse("geo:-34.903735,-57.938081");  
startActivity(i2);
```



Modificando **InfoActivity** para que pueda iniciarse desde otra app

- Vamos a crear un filtro de *intents* adecuado para la *activity* **InfoActivity** para que otra aplicación pueda iniciarla
- Vamos a definir nuestra propia acción con el string "GESTION_INFO"
- Luego crearemos otra aplicación que utilizará un Intent implícito para solicitar a Android que inicie la activity que pueda realizar la función "GESTION_INFO"

Modificando **InfoActivity** para que pueda iniciarse desde otra app

En **androidManifest.xml** buscar la etiqueta correspondiente a **infoActivity** y agregar el siguiente filtro:

Modificar el atributo “**exported**” declarado en la actividad **InfoActivity** con un valor “**true**” para que permita llamar a esa actividad desde otra aplicación.

```
<activity  
    android:name=".InfoActivity"  
    android:exported="true">  
    <intent-filter>
```

Correr en el emulador para actualizar la app en el dispositivo virtual

```
        <action android:name="GESTION_INFO" />
```

```
        <category android:name="android.intent.category.DEFAULT" />
```

```
    </intent-filter>
```

Debe incluir la categoría **CATEGORY_DEFAULT** en el filtro de *intents* porque los métodos **startActivity()** y **startActivityForResult()** tratan todos los *intents* como si pertenecieran a la categoría **CATEGORY_DEFAULT**. Si no declara esta categoría en el filtro de *intents*, no se aplicará ningún intent implícito a la *activity*

Creando una nueva aplicación

Crear una nueva aplicación basada en la plantilla

Empty Views Activity y definir la siguiente interfaz

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

Creando una nueva aplicación

Codificar el método **verInfo()** de la siguiente manera:

```
fun verInfo(v: View) {  
    val i = Intent();  
    i.action= "GESTION_INFO";  
    i.putExtra("dato1", "El año mostrado es");  
    i.putExtra("dato2", 2023);  
    startActivity(i);  
}
```

Correr en el
emulador