Trabajando con Múltiples Actividades

En esta parte del curso, veremos algunas formas de cómo trabajar con pantallas múltiples. Algunas aplicaciones pueden no necesitar más de una actividad, pero algunas aplicaciones pueden requerir varias. Pronto, sin dudas, encontrarás la necesidad de trabajar con múltiples actividades. No es particularmente difícil trabajar con aplicaciones de pantallas múltiples, pero tenemos que retroceder un poco para considerar cómo se diseñan las aplicaciones de Android.

Activación de componentes

La plataforma Android es genial en acoplamiento flexible. Una aplicación no es más que una colección de componentes unidos por un archivo de manifiesto, y cada uno de estos componentes se puede activar enviándole un mensaje. Si deseas mostrar (activar) una Actividad (Activity), necesitas crear un mensaje, enviarlo al tiempo de ejecución y dejar que el tiempo de ejecución lo active por ti.

No puedes tratar un componente directamente. El listado 9.1 muestra un pseudocódigo sobre cómo otras plataformas pueden mostrar una segunda pantalla: muchos desarrolladores están bastante familiarizados con este lenguaje, pero desafortunadamente no funcionará en Android

```
class FirstActivity extends AppCompatActivity
  implements View.OnClickListener {

  public void onClick(View v) {
    SecondActivity second = new SecondActivity(); // WON'T WORK
  }
}

class SecondActivity extends AppCompatActivity {
}
```

Listado 9.1: FirstActivity.java.

Para activar un componente, como una actividad, tenemos que hacer lo siguiente.

1. Crear un objeto Intent
2. Especificar qué queremos hacer o incluso cómo hacerlo
3. Enviar el objeto Intent al tiempo de ejecución y deja que se encargue de la activación de los componentes

Los intentos pueden activar componentes en la plataforma Android; son un mecanismo de transmisión de mensajes que puedes usar si deseas trabajar con una actividad, servicio, proveedor de contenido o receptor de difusión.

Los intentos tienen un papel muy importante en la plataforma de Android, y sus capacidades van más allá del simple lanzamiento de otra actividad. Algunas de las capacidades de Intents pueden estar un poco avanzadas para un curso para principiantes y, por lo tanto, no se analizarán aquí. Pero en este tema, veremos un par de cosas interesantes sobre los propósitos y las actividades. Por ejemplo:

- Lanzar otra actividad usando un intento explícito
- Pasar datos de una actividad a otra
- Devolver datos de una segunda actividad a la actividad principal
- Ciclo de vida de los métodos de actividades
- Un poco de Fragmentos

Un intento implícito es como preguntarle a alguien (en nuestro caso, este es el tiempo de ejecución de Android) para comprar un poco de azúcar. Realmente no importa de dónde lo saca: podrías ir a la tienda de conveniencia cercana o comprarla en una supertienda al otro lado de la ciudad.

Realmente no nos importa mientras tengamos el azúcar. Una intención explícita, por otro lado, es pedirle a alguien que nos dé algo de azúcar, y él tiene que comprarla a 7/11 a 3 cuadras de distancia. Volviendo a nuestro pseudocódigo (Listado 9.1), si sabemos que queremos lanzar una actividad específica (SecondActivity), podemos usar un intento explícito. El siguiente pseudocódigo muestra cómo lograr esto.

```
// FirstActivity.java
class FirstActivity extends AppCompatActivity
  implements View.OnClickListener {

  public void onClick(View v) {
    Intent intent = new Intent(this, SecondActivity.class);
    startActivity(intent);
  }
}

// SecondActivity.java
class SecondActivity extends AppCompatActivity {

}
```

Las siguientes secciones entrarán en detalles que ilustran cómo trabajar con actividades e intenciones.

Lanzar una actividad específica

Cuando deseas iniciar otra actividad (o cualquier componente), deberás usar intenciones explícitas. Este objeto de intención se crea y se inicia (generalmente) en la actividad que desea iniciar la activación; este es un proceso de dos pasos.

En primer lugar, debemos crear el objeto de intención.

Intent intennt = new Intent (<Context>, <Target>);

El objeto Context es simplemente una referencia al estado del componente que desea iniciar o lanzar la intención: esta suele ser la palabra clave this o, si está creando la intención desde dentro de una clase interna, será ActivityName.this (donde ActivityName es el nombre de tu actividad, por ejemplo, MainActivity).

El objetivo es el nombre de la Activity que deseas lanzar; esto generalmente se escribe NameOfActivity.class.

Después de que se haya creado el objeto intencionado, ahora se puede iniciar con el siguiente comando:

startActivity (intención);

Si, por ejemplo, queremos lanzar una actividad llamada SecondActivity desde MainActivity.java, podemos gestionar esto con el siguiente código:

```
Intent intent = new Intent (MainActivity.this, SecondActivity.class);
startActivity (intent);
```

En este punto, el tiempo de ejecución de Android habría resuelto el intento, y si encuentra SecondActivity.class, se abrirá. MainActivity se desenfocará y no será visible para el usuario porque SecondActivity ocupará toda la pantalla del dispositivo.

El usuario solo podrá navegar de regreso a MainActivity si (1) el usuario usa el botón Atrás, (2) SecondActivity termina, o (3) un botón de retorno está codificado en SecondActivity que llama a MainActivity usando otro objeto Intent (que no lo haremos en este proyecto).

Reunamos todas estas cosas en un proyecto de demostración.

Proyecto Demo

En esta sección veremos uno de los usos fundamentales de un objeto Intent. Lanzaremos una subactividad (segunda actividad) de la actividad principal.

1. Crea un nuevo proyecto
2. Elimina el textView generado de activity_main y coloca una vista de botón en su lugar. Utilizaremos el botón para iniciar la segunda actividad, la llamaremos subactividad
3. Crea una nueva Actividad vacía en el menú Archivo AS3
4. Coloca un botón en el segundo diseño de actividad. Usaremos este botón para apagar o cerrar la actividad
5. Ir a la actividad principal, crear el controlador, crear la intención, encender el AVD, iniciar la segunda actividad
6. Ir a la segunda actividad
7. Crear un botón, crear un controlador, implementar la función (finish())

Crea un nuevo proyecto usando la siguiente información (ver la Tabla 9.1).

Application Name	**FirstIntent**
Company domain	Usa tu sitio web o inventa algo; recuerda que esto está en la notación DNS inversa.
Project location	Deja el valor predeterminado. Ignora el soporte de C++ y Kotlin.
Form factor	Solo teléfono y tableta.
Minimum SDK	API 26 Oreo
Type of activity	Vacío (Empty)
Activity name	MainActivity
Layout name	activity_main.

Tabla 9.1: Detalles del proyecto para FirstIntent.

Ve a actvity_main.xml en la ventana del editor principal y elimina el objeto textView generado (Hello).

Coloca una vista de Botón en el diseño haciendo clic y arrastrando la vista de Botón desde la paleta.

Coloca el botón donde desees y haz clic en “Inferir restricciones” en el inspector de restricciones como se muestra en la figura 9.1.

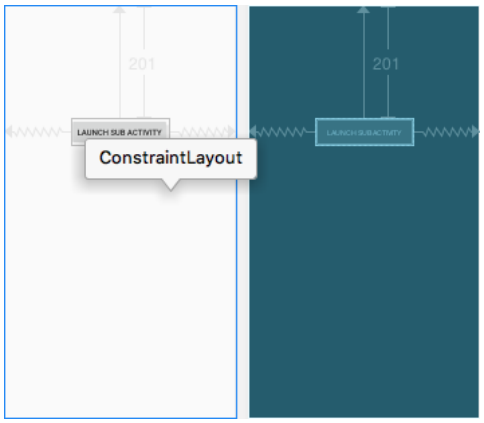

Figura 911: activity_main con Button view.

Cambia el texto (etiqueta) del Botón: puedes hacerlo editando el activity_main. xml directamente o cambiando el atributo de texto del Botón en el inspector de atributos (consulta la figura 9.2).

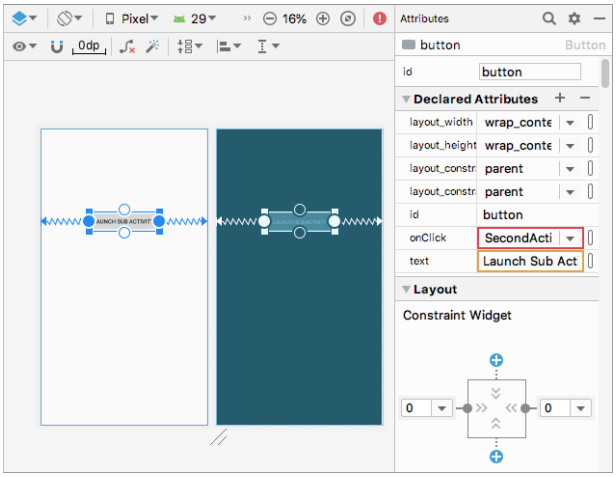

Figura 9.2: Inspector de atributos.

Alternativamente, puedes cambiar al "modo de texto" y editar el archivo de diseño directamente como se muestra en el Listado 9.2.

```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="201dp"
    android:onClick="SecondActivity"
    android:text="Launch Sub Activity"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Listado 9.2: Edición de la etiqueta de la vista de botones.

(1) El atributo android: text es la etiqueta de la vista Button. Esto es lo que el usuario verá escrito en el Botón.
Lo siguiente es crear otro componente de actividad. Puedes hacer esto seleccionando la carpeta "app" en la ventana de la herramienta del proyecto (como se muestra en la figura 9.3), luego Nueva ➤ Actividad ➤ Actividad vacía.

Del mismo modo, también puedes lograr lo mismo yendo a la barra del menú principal Archivo ➤ Actividad ➤ Actividad vacía; solo asegúrate de que, en cualquier caso, hayas seleccionado la carpeta "app" en la ventana de la herramienta del proyecto.

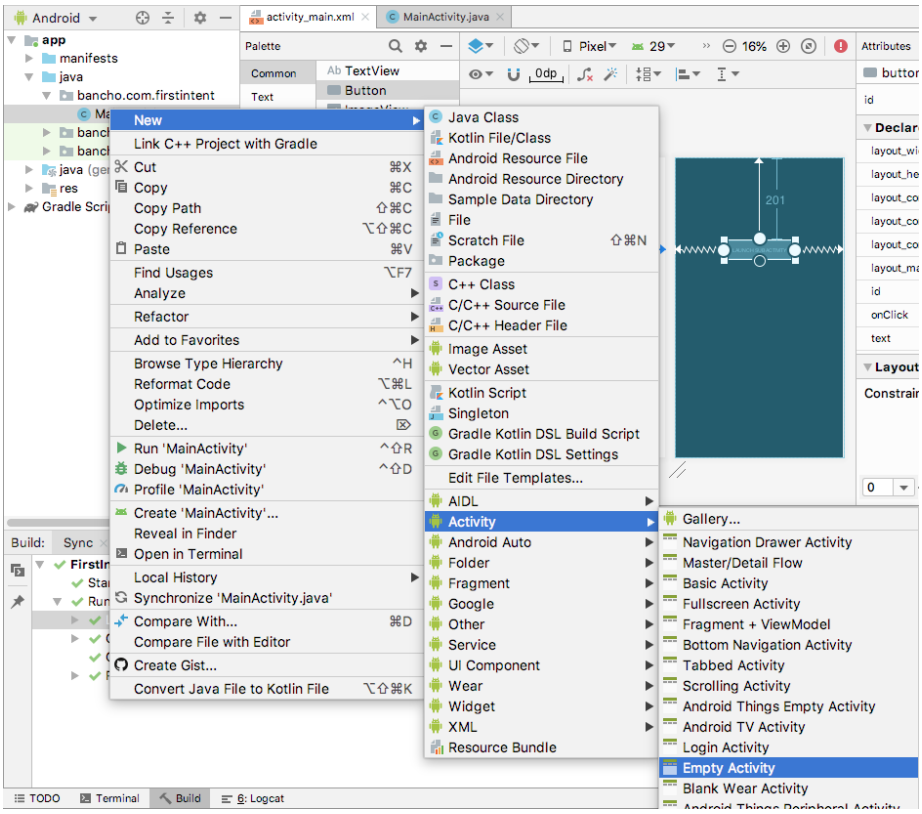

Figura 9.3: Nueva Actividad.

La figura 9.4 muestra el diálogo de creación para el segundo componente de actividad.

Cambia el nombre a SecondActivity y el nombre del diseño a activity_second; aceptaremos el valor predeterminado para el nombre del paquete porque lo queremos en el mismo paquete que la actividad principal.

Nota. Todos los componentes de la actividad en una aplicación deben estar registrados en el archivo AndroidManifest.

Una de las ventajas de utilizar el asistente de actividades es que también actualiza automáticamente el archivo AndroidManifest, agregando así una declaración para la actividad recién creada.
Puedes ver el AndroidManifest.xml en app/manifest/AndroidManifest.xml.

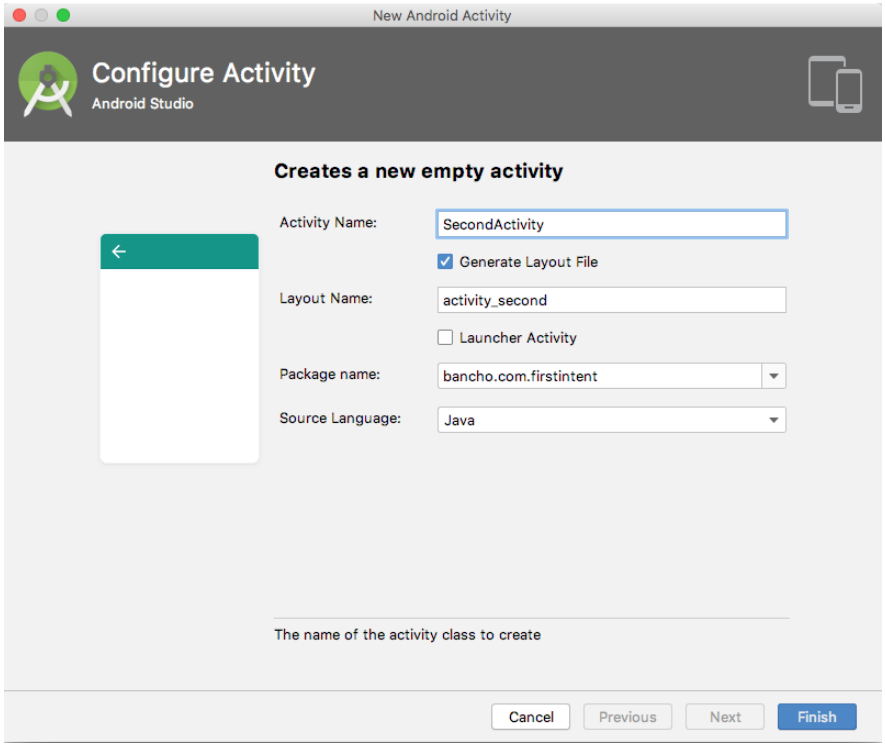

Figura 9.4: Segunda Actividad.

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".SecondActivity"></activity> 1
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

Listado 9.3: Entrada de SecondActivity en el AndroidManifest.

(1) El asistente de actividades agregó esta declaración al AndroidManifest. Si no hubiéramos usado el asistente para crear la segunda actividad, será nuestra responsabilidad agregar manualmente esta entrada al manifiesto: esta es una buena razón para usar siempre asistentes cuando hay uno disponible.

Ahora que se ha creado la segunda actividad, podemos agregar el código de gestión de eventos a nuestra actividad principal. Como recordarás en el último tema, hay dos maneras de agregar capacidad de manejo de eventos a nuestra aplicación; mientras que podemos hacerlo programáticamente o declarativamente, elegiremos el último.

Abre el archivo de diseño principal (activity_main) en modo de diseño, selecciona el botón y establece su atributo “onClick” en el valor “launchSecondActivity” (puedes hacer esto en el inspector de atributos mientras se selecciona el botón: consulta la figura 9.5).

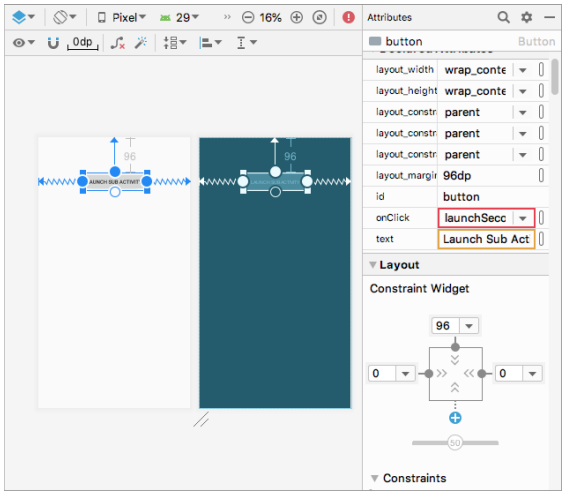

Figura 9.5: FirstActivity, atributos del inspector.

```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="96dp"
    android:onClick="launchSecondActivity"
    android:text="Launch Sub Activity"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    tools:layout_editor_absoluteY="96dp"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

(1)

Listado 9.4: activity_main con el controlador onClick.

(1) Cuando estableces el atributo onClick de la vista Button en el inspector, el archivo de diseño se actualizará automáticamente. Por el contrario, puedes ir directamente a este archivo de diseño y agregar el atributo onClick como se ve en este ejemplo de código

El siguiente paso es implementar realmente el método launchSecondActivity en el archivo de actividad asociado (MainActivity.java).

TWO PARTS OF AN ACTIVITY

Recuerda que un componente de actividad tiene dos partes, un archivo de diseño (xml) y un archivo de programa (Java). Ahora que tenemos dos componentes de actividad, tenemos un total de cuatro archivos para trabajar:

Layout file	Program file
activity_main.xml	MainActivity.java
activity_second.xml	SecondActivity.java

Si hay alguna acción que desees que ocurra como resultado de un evento generado por el usuario en activity_main, debes codificar eso en MainActivity.java. De forma similar, si el evento se desencadena desde activity_second, ese código entra en SecondActivity.java.

```
package bancho.com.firstintent;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button objButton = (Button) findViewById(R.id.button);
    }

    public void launchSecondActivity(View) {  1
        Intent i = new Intent( packageContext: this, SecondActivity.class);  2 3 4
        startActivity(i);  5
    }
}
```

Listado 9.5: Código de manejo de eventos para launchSecondActivity.

(1) Al igual que en el último tema, asegúrate de que el nombre del método esté escrito exactamente como está escrito en el atributo onClick de la vista Button. Además, asegúrate de que el método adopte un argumento de Vista.

(2) Se crea un objeto Intento y se le asigna a una variable. El constructor de intención toma dos argumentos.

(3) El primer argumento es un objeto de contexto; generalmente se refiere al objeto que quiere iniciar o lanzar el intento. La palabra clave this se usa porque queremos lanzar la intención desde MainActivity, y dado que el método de manejo de eventos (launchSecondActivity) es un miembro de MainActivity, podemos simplemente usar la palabra clave this -si quieres ser preciso, podrías sustituir la palabra clave this con la más detallada y explícita MainActivity.this.

(4) El segundo argumento del constructor de intención es la clase de componente de la actividad que realmente deseas iniciar. Esto generalmente se construye como el nombre del archivo de programa adjunto con .class, que hace referencia a la versión del código de bytes de Java (compilado) de la clase; por lo tanto, en este caso, es SecondActivity.class. Este es un ejemplo de un intento explícito porque el nombre exacto de la actividad objetivo del intento está codificado (hard-coded). Los intentos implícitos, por el contrario, no especifican la actividad exacta que quiere lanzar; en cambio, se basa en el tiempo de ejecución de Android para resolver la intención para que se pueda poner en marcha el componente que mejor se adapte a el.

(5) Para lanzar finalmente la segunda actividad, llamamos al método startActivity pasar la intención como argumento. No puedes simplemente crear una instancia de la clase SecondActivity para iniciarlo. Recuerda que una aplicación de Android es un conjunto de componentes débilmente acoplados que se mantienen unidos por al menos dos cosas, a saber, AndroidManifest e Intents. El método startActivity simplemente le dice al tiempo de ejecución de Android que queremos activar otro componente y que estamos utilizando un objeto de intención para ayudar a que el tiempo de ejecución resuelva la solicitud.

Si ejecutamos esta aplicación ahora, lo que veremos es la actividad principal con un solo botón; cuando se hace clic en el botón, la pantalla cambiará y mostrará la segunda actividad. Ese comportamiento debería estar bien, pero para completar realmente el ejercicio, agregaremos un botón a la segunda actividad que funcionará como un botón “cerrar”.

Cuando se hace clic en él, debe cerrarse y eliminar la segunda actividad, lo que lo eliminará en la pila de la pantalla, haciendo que nuestra actividad principal vuelva a ser visible para el usuario.

Ve al archivo de diseño de la segunda actividad (activity_second.xml) y añádale una vista de Botón (consulta la Figura 6-5). Como nuestra actividad principal, mantenlo simple; coloca el botón donde deseas que aparezca y usa las "Restricciones de Infer" en el inspector de restricciones para un diseño fácil y automático.

Establece el atributo de texto del Botón en "Cerrar" (Figura 6-6) pero deja en blanco el controlador onClick. Manejaremos el evento programáticamente en el archivo asociado de Java.

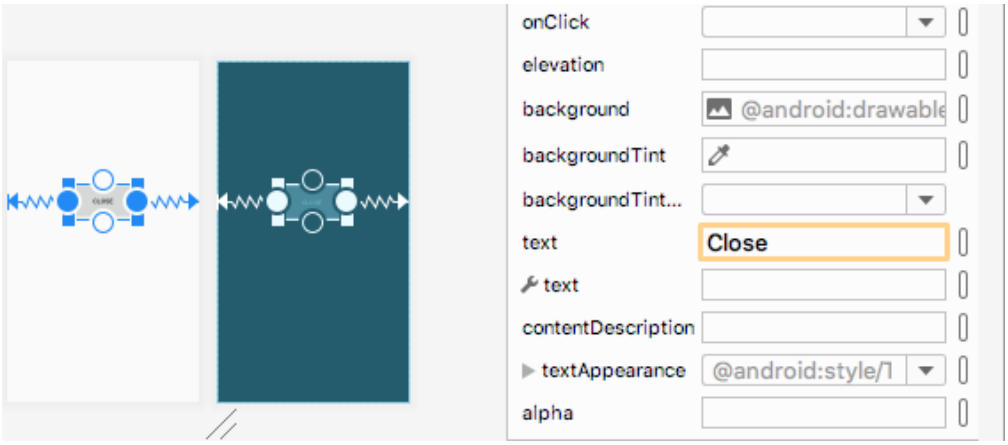

Figura 9.6: activity_second con el botón Close.

```
<?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=".SecondActivity">

    <Button
        android:id="@+id/button2"  (1)
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="68dp"
        android:text="Close"  (2)
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:layout_editor_absoluteY="68dp"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Figura 9.6: Botón en activity_second.

(1) Si no cambiaste el atributo id del botón, será button2, ya que esta es la segunda vista Button en todo el proyecto. Puedes cambiar este valor aquí en el archivo de diseño o en el inspector de atributos, pero lo dejaremos como predeterminado; button2 debería estar bien.

(2) Cambia la etiqueta de texto del Botón a "Close" para que sea un poco descriptivo. De nuevo, esto se puede hacer directamente aquí en el archivo XML o en el inspector de atributos (Ver la figura 9.5).

Lo último que debes hacer antes de probar la aplicación es implementar algún código de manejo de eventos en respuesta al clic del botón. Consulta el Listado 9.7 para obtener el código completo de SecondActivity.java.

```
package bancho.com.firstintent;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class SecondActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second); 1

        Button secondButton = (Button) findViewById(R.id.button2); 2
        assert secondButton != null; 3
        secondButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                finish(); 4
            }
        });
    }
}
```

Listado 9.7: Código completo de SecondActivity.java.

(1) Ya sabemos lo que hace esto; este es el código de pegamento que asocia este archivo Java a un archivo de diseño. El tiempo de ejecución inflará el archivo de diseño XML, actualizará la clase R y creará los objetos de Java reales que representan los objetos de vista del archivo de diseño, lo que a su vez hará que estos objetos de Java estén disponibles para que hagamos referencia en nuestro código. Comprende la "pantalla" para el objeto Activity.

(2) findViewById es un método de localización; intenta encontrar el objeto Java que se creó durante el proceso de inflación. Si se encuentra, la dirección de ese objeto se almacenará en la variable llamada secondButton.

(3) Esto es solo codificación defensiva; solo nos estamos asegurando de que secondButton realmente contenga la dirección de un objeto, y que no sea nulo o esté vacío.

(4) Este es el código importante. Cuando se llama al método de finalización contra un objeto Activity, efectivamente destruye la actividad y la cierra. Destruir la Actividad lo quita de la memoria.

Pasar datos a otra actividad

En esta sección, exploraremos cómo pasar datos de la actividad principal a una segunda actividad. Mientras hacemos esto, practicaremos un poco de programación matemática. Intentaremos calcular el MCD (el factor común más grande) de dos números. Crearemos dos actividades (MainActivity y Calculate). MainActivity hará lo siguiente:

1. Espera a que el usuario ingrese (dos números), así crearemos dos objetos de vista de texto sin formato.
2. Restringir las entradas a solo dígitos; no tiene sentido aceptar entradas alfanuméricas.
3. Comprueba si los campos de texto están vacíos; solo queremos proceder si están correctamente llenos con los números.
4. Crea un intento, y luego nos aprovecharemos de eso para que podamos obtener los dos números ingresados para la actividad Calcular.

La segunda actividad (Calcular, Calculate) es el caballo de batalla. Será la que hará el cálculo numérico. Aquí hay un desglose de sus tareas:

1. Obtener la intención que se pasó de MainActivity.
2. Comprueba si hay algunos datos piggybacking en él.
3. Si hay datos, vamos a extraerlos para que podamos usarlo para el cálculo.
4. Cuando el cálculo es hecho, mostraremos los resultados en un objeto de vista de texto.

Acerca del algoritmo para el MCD

Hay muchas maneras de calcular el MCD, pero el más conocido es probablemente el algoritmo de Euclides.

Lo implementaremos de esta manera.

1. Obtén la entrada de dos números.
2. Encuentra el número más grande.
3. Divide el número más grande usando el número más pequeño.
    - Si el resto del paso no. 3 es cero, entonces el MCD es el número más pequeño
    - Por otro lado, si el resto no es cero, haz lo siguiente:
        - Asigna el valor del número más pequeño al número más grande, luego asigna el valor del resto al número más pequeño
        - Repita el paso no. 3 (hasta que el resto sea cero)

Vamos a crear un nuevo proyecto para el ejercicio MCD; ver la Tabla 9.2 para más detalles.

Application Name	MCD
Company domain	Usa tu sitio web o inventa algo; recuerda que esto está en la notación DNS inversa.
Project location	Deja el valor predeterminado.
Form factor	Solo teléfono y tableta.
Minimum SDK	API 26 Oreo
Type of activity	Vacío (Empty)
Activity name	MainActivity (defecto)
Layout name	activity_main (defecto)

Tabla 9.2: Detalles del proyecto MCD.

Este proyecto tendrá una segunda actividad. Cuando el proyecto haya terminado de crear la actividad y Gradle se haya completado en la compilación, agrega la segunda actividad. En la ventana de la herramienta del proyecto, haz clic con el botón derecho en la aplicación ➤ Actividad ➤ Actividad vacía. Usa los detalles en la Tabla 9.3 para la actividad recién creada.

Activity name	Calculate
Layout name	activity_calculate

Tabla 9.3: Second Activity.

Las Figuras 9.7 y 9.8 muestran el diseño de los archivos para activity_main y activity_calculate.

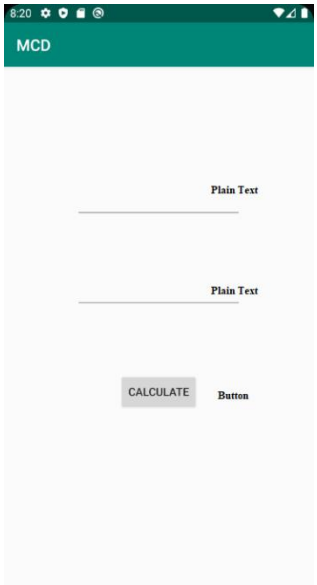

Figura 9.7: diseño activity_main.

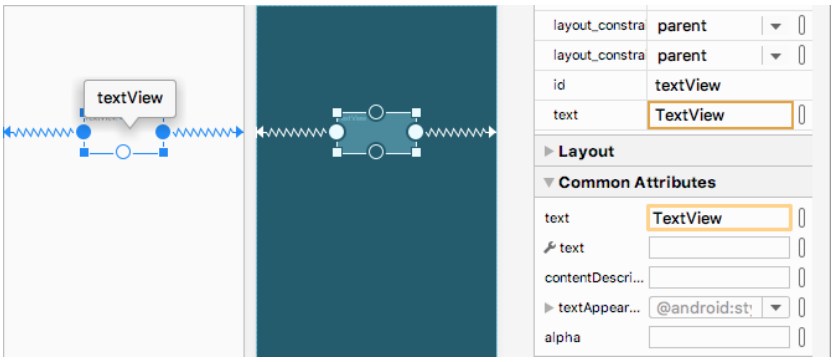

Figura 9.8: diseño activity_calculate.

```xml
<?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">

    <EditText
        android:id="@+id/firstno"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="63dp"
        android:ems="10"
        android:gravity="center_vertical|center_horizontal" (1)
        android:inputType="number" (2)
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <EditText
        android:id="@+id/secondno"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="28dp"
        android:ems="10"
        android:gravity="center"
        android:inputType="number"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/firstno" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="29dp"
        android:text="CALCULATE"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/secondno" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Listado 9.8: activity-main.xml.

(1) Establece el atributo de gravedad para centrar vertical y horizontal para alinear el texto al centro.

(2) Esto restringe la entrada solo a números.

Tip:

1. Puedes configurar el atributo de gravedad de EditText en el inspector de atributos. Mientras EditText está seleccionado en el editor de modo de diseño, haz clic en el botón “Ver todos los atributos” en el inspector, como se muestra en la figura 9.9.
2. Puedes establecer el tipo de entrada de EditText en el inspector de atributos haciendo clic en los puntos suspensivos (...) al lado de “tipo de entrada”. Las opciones para los tipos de entrada serán visibles en una ventana emergente (vea la figura 9.10).

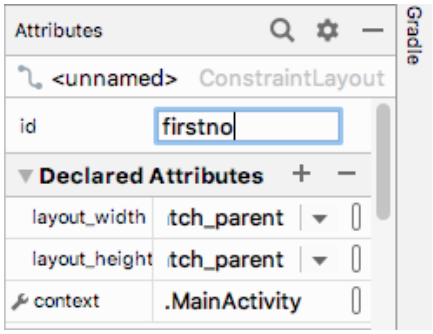

Figura 9.9: Ver todos los atributos en el inspector.

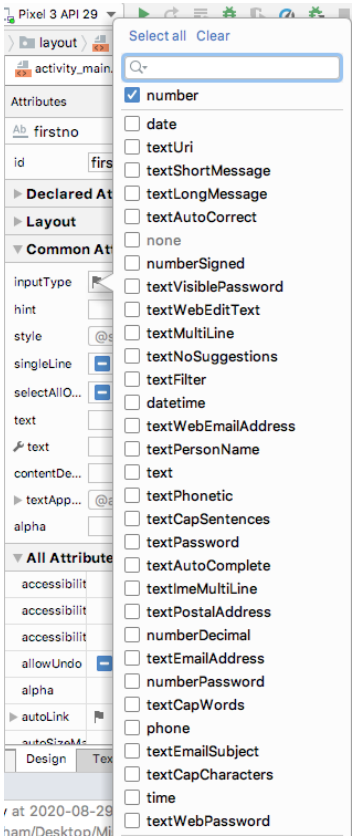

Figura 9.10: Atributo de tipo de entrada en el inspector.

Ahora podemos pasar a los detalles de diseño de activity_calculate. Los elementos de la interfaz de usuario de la segunda actividad son muy simples; solo hay un elemento TextView. El archivo XML del activity_calculate se muestra en el Listado 9.9.

```
<?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=".Calculate">

    <TextView
        android:id="@+id/textView"
        android:layout_width="176dp"
        android:layout_height="76dp"
        android:layout_marginTop="113dp"
        android:gravity="center"
        android:text="TextView"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Listado 9.9: Archivo de diseño de la actividad Calcular.

Los detalles sobre cómo establecer las restricciones ya se te dejarán a ti. Ya hemos visto ejemplos detallados sobre cómo trabajar con el diseño de restricciones. La forma más rápida de tener un diseño decente es hacer lo siguiente:

1. Arrastra y coloca cada objeto de visualización en la ubicación aproximada donde deseas que se muestren.
2. Utiliza las herramientas “empacar” y “alinear2 en el inspector de restricciones (consulta figura 9.11).

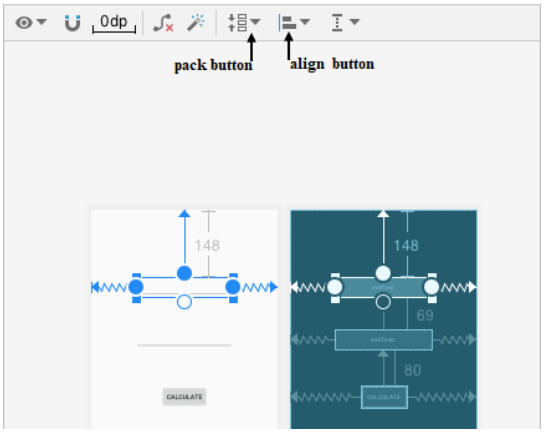

Figura 9.11: Inspector de restricciones.

Ahora que tenemos un diseño de interfaz de usuario básico, echemos un vistazo a cómo podemos escribir códigos que accedieron a estos elementos de la interfaz de usuario.
La siguiente lista muestra la estructura básica de MainActivity. Las variables fno, sno se definen como miembros de la clase porque los referenciaremos desde los métodos onClick y

onCreate. MainActivity es el objeto oyente; es por eso que el método onClick está anulado en el cuerpo de la actividad principal.

```
package bancho.com.mcd;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.text.TextUtils;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    private EditText fno;
    private EditText sno;
    private Button btn;

    @Override
    protected void onCreate(Bundle savedInstanceState) {...}

    @Override
    protected void onStart(){...}

    public void onClick(View v){...}
}
```

Listado 9.10: MainActivity.java.

Nota. A medida que comienzas a escribir los códigos, AS4 podría indicar que hay advertencias y errores al mostrar algunos iconos de bulbo y líneas onduladas. Lo más probable es que esto se deba a que se han anulado las declaraciones de importación y el método (aún) que faltan. Utiliza la solución rápida para resolver estas advertencias y errores: (Alt + Intro para Windows y Linux | Opción + Intro para macOS).

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    fno = (EditText) findViewById(R.id.firstno);
    sno = (EditText) findViewById(R.id.secondno);
    btn = (Button) findViewById(R.id.secondno);
    btn.setOnClickListener(this);
}
```

Listado 9.11: Método onCreate.

Este es un código muy típico para el método onCreate. La mayoría de nuestras muestras de código se verán casi idénticas a esto.

```
public void onClick(View v){
    boolean a = TextUtils.isEmpty(fno.getText());
    boolean b = TextUtils.isEmpty(sno.getText());

    if (!a & !b) {

        int firstnumber = Integer.parseInt(fno.getText().toString());
        int secondnumber = Integer.parseInt(sno.getText().toString());

        Intent intent = new Intent( packageContext: this, Calculate.class);
        Bundle bundle = new Bundle();
        bundle.putInt("fno", firstnumber);
        bundle.putInt("sno", secondnumber);
        intent.putExtra( name: "gcfdata", bundle);
        startActivity(intent);
    }
}
```

Listado 9.12: Método onClick.

(1) Text TextUtils puede verificar si un objeto TextView no tiene ningún texto dentro. Puedes buscar un campo de texto vacío de otra forma extrayendo la cadena dentro de él y verificando si la longitud es mayor que cero, pero TextUtils es una forma más concisa de hacerlo.

(2) Asegúrate de que ambos campos de texto no estén vacíos. Si uno de ellos está vacío, todos los códigos dentro de este bloque serán simplemente evadidos, por lo que no hay daño ni falta. La aplicación esperará diligentemente la entrada del usuario.

(3) El método getText () devuelve un objeto Editable, que no es compatible con el método parseInt de la clase Integer. El método toString debe convertir el objeto Editable en una cadena (String) normal.

(4) Esta línea crea un objeto Intent. El primer argumento para el constructor Intent es un objeto de contexto. El intento necesita saber desde dónde se está lanzando, de ahí la palabra clave this; estamos lanzando la intención de nosotros mismos (MainActivity). El segundo argumento para el constructor es la actividad de destino que queremos lanzar.

(5) Vamos a incorporar algunos datos en el objeto de intención, por lo que necesitaremos un contenedor para estos datos. Un objeto Bundle es como un diccionario; almacena datos en pares llave/valor.

(6) El objeto Bundle es compatible con una gran cantidad de métodos put que se ocupan de completar el paquete. El paquete puede almacenar una variedad de datos, no solo enteros. Si quisiéramos insertar una cadena (string) en el Bundle, podríamos decir bundle.putString() o bundle.putBoolean() si quisiéramos almacenar datos booleanos.

(7) Después de haber poblado el objeto Bundle, ahora podemos utilizar el objeto Intent llamando al método putExtra. Similar al objeto Bundle, Intent también usa el par de llave/valor para poblar y acceder a los extras.
En este caso, "gcfdata". Necesitamos usar la misma llave más tarde (en la segunda actividad) para recuperar el paquete.

(8) Esta declaración lanzará la Actividad.

```
@Override
protected void onStart(){
    super.onStart();
    fno.setText("");
    sno.setText("");
}
```

Listado 9.13: Método onStart.

El método onStart de MainActivity se puede llamar muchas veces en el ciclo de vida de la aplicación. Se ejecutará por primera vez cuando ejecutemos la aplicación (después de onCreate), y posteriormente cada vez que otra actividad capte el foco y luego el usuario navegue de regreso a MainActivity.
Cada vez que eso sucede, simplemente borramos el contenido de los campos de texto. El listado 9.14 muestra el código completo para MainActivity.

MainActivity.java es en su mayoría código repetitivo. Solo se ocupa de la entrada y del lanzamiento de la segunda actividad. El trabajo real del GCF (MCD) ocurre dentro de la actividad Calculate. Veamos el código de Calculate.java. Después de que MainActivity transfiera algunos datos usando los objetos Intent y Bundle, las primeras cosas que debemos cuidar dentro de la actividad Calculate es extraer esos datos del paquete y eventualmente extraer los pares llave/valor de datos dentro del paquete.

```
Intent intent = getIntent();                          (1)
Bundle bundle = intent.getBundleExtra("gcfdata");     (2)
```

(1) Este código se llamará dentro del método onCreate de la actividad Calculate; la instrucción getIntent aquí devolverá el objeto de intención que se utilizó para iniciar esta actividad.
(2) getBundleExtra devuelve el objeto del paquete que pasamos al objeto intent en MainActivity. Recuerda que cuando insertamos el objeto bundle en MainActivity, usamos la clave "gcfdata"; por lo tanto, necesitamos usar la misma llave aquí para extraer el paquete.

Una vez que hemos extraído el paquete con éxito, podemos obtener los dos valores enteros que guardamos en él antes.

```
package bancho.com.mcd;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.text.TextUtils;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    private EditText fno;
    private EditText sno;
    private Button btn;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        fno = (EditText) findViewById(R.id.firstno);
        sno = (EditText) findViewById(R.id.secondno);
        btn = (Button) findViewById(R.id.secondno);
        btn.setOnClickListener(this);
    }

    @Override
    protected void onStart(){
        super.onStart();
        fno.setText("");
        sno.setText("");
    }

    public void onClick(View v){
        boolean a = TextUtils.isEmpty(fno.getText());
        boolean b = TextUtils.isEmpty(sno.getText());

        if (!a & !b) {

            int firstnumber = Integer.parseInt(fno.getText().toString());
            int secondnumber = Integer.parseInt(sno.getText().toString());

            Intent intent = new Intent(this, Calculate.class);
            Bundle bundle = new Bundle();
            bundle.putInt("fno", firstnumber);
            bundle.putInt("sno", secondnumber);
            intent.putExtra("gcfdata", bundle);
            startActivity(intent);
        }
    }
}
```

Listado 9.14: MainActivity.

int first = bundle.getInt ("fno", 1);
int second = bundle.getInt ("sno", 1);

El primer parámetro del método getInt es simplemente la llave. Esta tiene que ser la misma llave que usamos en el método putInt (de vuelta en MainActivity). El segundo parámetro opcional es simplemente un valor predeterminado, en caso de que la clave no se encuentre en el paquete. Los próximos pasos serán comenzar a calcular el MCD.

```
int bigno, smallno = 0;
int rem = 1;

if (first > second) {    (1)
    bigno = first;
    smallno = second;
}
else {
    bigno = second;
    smallno = first;
}
while ((rem = bigno % smallno) != 0) {    (2)
    bigno = smallno;
    smallno = rem;
}
gcftext.setText(String.format("MCD =  %d", smallno));    (3)
```

Listado 9.15: Lógica MDC.

(1) Intentamos averiguar cuál es el número más grande, una simple declaración if y algunas asignaciones a la variable bigno y smallno deberían encargarse de ello.

(2) Hay dos cosas en esta declaración. Primero, dividimos bigno con smallno y asignamos el resto a la variable rem.
A continuación, toda esta expresión se está probando para determinar si el resultado es cero, porque si lo es, deberíamos salir del ciclo while. Significa que ya hemos encontrado el GCF (MDC). Si no es igual a cero, cambia los valores de bigno y smallno según el algoritmo de Euclides.

(3) Una vez que encontramos el GCF, estableceremos su valor como el texto del objeto TextView.

Las Figuras 9.12 y 9.13 muestran la aplicación de MCD en acción.

```java
package bancho.com.mcd;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.widget.TextView;

public class Calculate extends AppCompatActivity {

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

        int bigno, smallno = 0;
        int rem = 1;

        TextView gcftext = (TextView) findViewById(R.id.textView);
        Intent intent = getIntent();
        Bundle bundle = intent.getBundleExtra("gcfdata");

        if ((bundle != null) & !bundle.isEmpty()){

            int first = bundle.getInt("fno", 1);
            int second = bundle.getInt("sno", 2);

            if (first > second) {
                bigno = first;
                smallno = second;
            }
            else {
                bigno = second;
                smallno = first;
            }
            while ((rem = bigno % smallno) != 0) {
                bigno = smallno;
                smallno = rem;
            }
            gcftext.setText(String.format("MCD =  %d", smallno));
        }
    }
}
```

Listado 9.16: Código completo para Calculate.java.

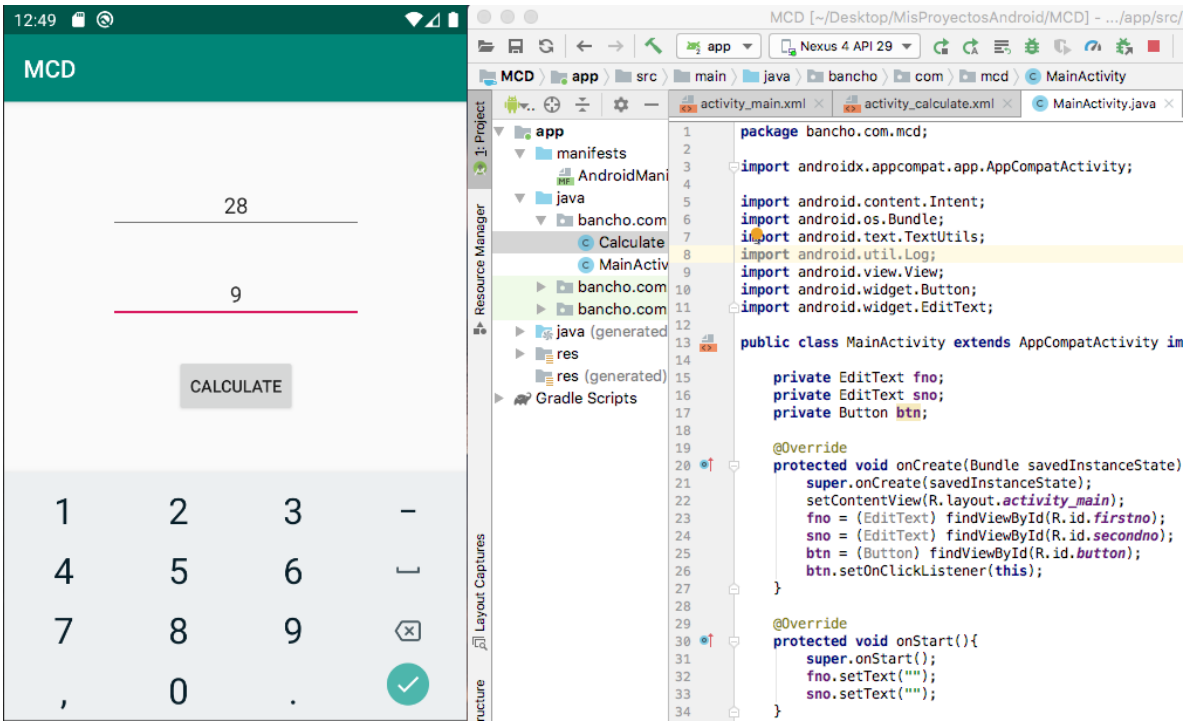

Figura 9.12: MCD MainActivity.

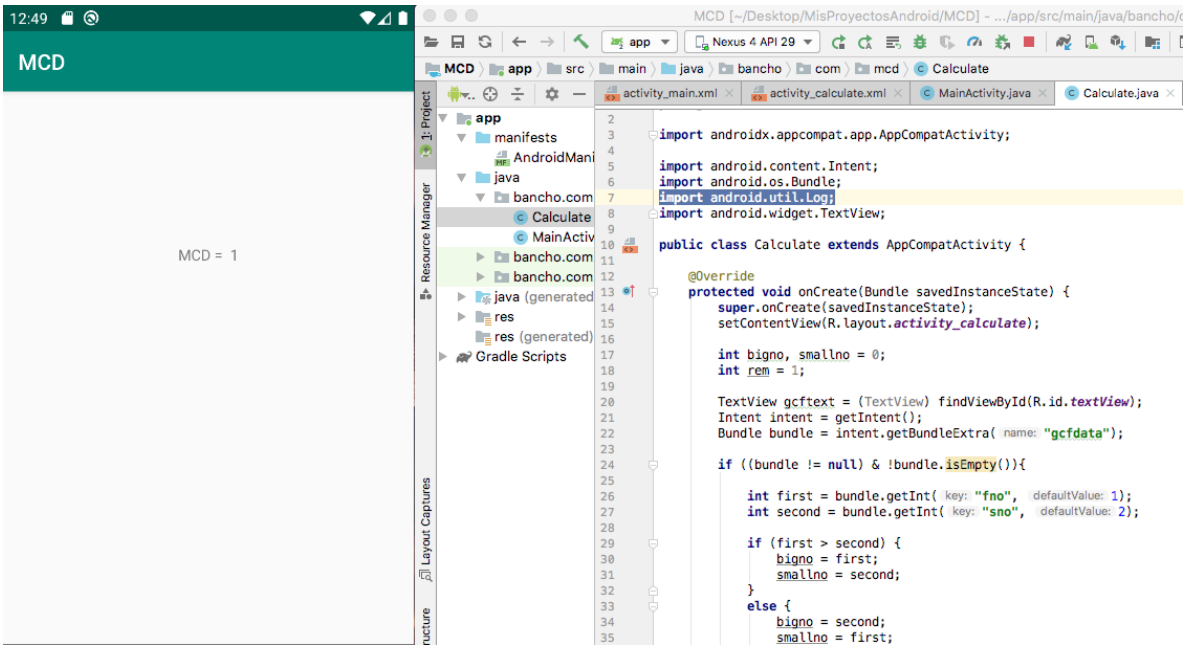

Figura 9.13: Resultado de MCD.

Devolución de resultados de otras actividades

En la sección anterior, lanzamos una subactividad y le pasamos algunos datos. En esta sección, veremos cómo devolver datos desde una subactividad. La figura 9.14 muestra la secuencia de eventos sobre cómo hacerlo.

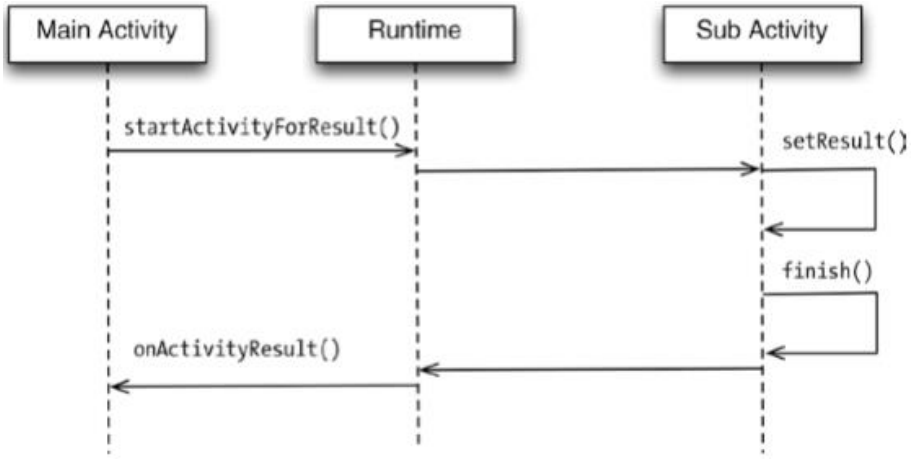

Figura 9.14: Secuencia de eventos.

Lanzaremos una subactividad desde MainActivity. Esto se puede gestionar creando un objeto de intención explícito y llamando a startActivityForResult. Una actividad puede lanzar otras múltiples actividades, y cada una de ellas podría arrojar algunos resultados. Cuando estos resultados vuelvan, todos estarán dentro del método onActivityResult, por lo que necesitamos saber de qué actividad proviene cada resultado; el REQUEST_CODE nos ayudará con eso.

```
Intent intent = new Intent(MainActivity.this, SubActivity.class);
startActivityForResult (intent, REQUEST_CODE)
```

Cuando el tiempo de ejecución resuelve el intento, se crea la Subactividad y se hará visible. Para entonces, podrás crear tus propios datos, tal vez a través de una entrada de usuario. Si deseas devolver los datos a MainActivity, necesitas crear un objeto de intención para enviar los datos de vuelta a MainActivity mediante piggybacking en el objeto intencionado.

```
String data = "Datos para enviar de vuelta";
intent.putExtra ("key", data);
setResult (Activity.RESULT_OK, intent);
finish();
```

Cuando SubActivity llama al método de finalización (finish), se destruirá y MainActivity volverá a la parte superior de la pila de actividades. El tiempo de ejecución llamará a

OnActivityResult de MainActivity; aquí es donde podemos extraer los datos que SubActivity envió. Vamos a configurar un proyecto de demostración para que podamos explorar estos conceptos y ver cómo se ven en el código.

Configuración del proyecto

Crea un nuevo proyecto usando los detalles en la Tabla 9.4.

Application Name	GetResultsSubActivity
Project location	Deja el valor predeterminado.
Form factor	Solo teléfono y tableta.
Minimum SDK	API 26 Oreo
Type of activity	Vacío (Empty)
Activity name	MainActivity (defecto)
Layout name	activity_main (defecto)

Tabla 9.4: Detalles para GetResultsSubActivity.

Este proyecto tendrá una segunda actividad. Cuando el proyecto haya terminado de crear la actividad y gradle se realice en la compilación, agrega la segunda actividad. En la ventana de la herramienta del proyecto, haz clic con el botón derecho en la aplicación ➤ Actividad ➤ Actividad vacía. Usa los detalles en la Tabla 9.5 para crear la nueva actividad.

Activity name	SecondActivity
Layout name	activity_second

Tabla 9.5: SecondActivity.

Figura 9.15: Elementos de la interfaz de usuario de las actividades.

```
<?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">

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="296dp"
        android:gravity="center"
        android:text="Launch 2nd Activity"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView" />

    <TextView
        android:id="@+id/textView"
        android:layout_width="211dp"
        android:layout_height="0dp"
        android:layout_marginBottom="40dp"
        android:layout_marginTop="80dp"
        android:gravity="center"
        android:text="TextView"
        app:layout_constraintBottom_toTopOf="@id/button"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Listado 9.17: activity:main.

```
<?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=".SecondActivity">

    <EditText
        android:id="@+id/editText"
        android:layout_width="262dp"
        android:layout_height="0dp"
        android:layout_marginBottom="27dp"
        android:layout_marginTop="90dp"
        android:ems="10"
        android:gravity="center"
        android:hint="Type something"
        android:inputType="text"
        app:layout_constraintBottom_toTopOf="@id/button2"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="301dp"
        android:gravity="center"
        android:text="Close"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@id/editText"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Listado 9.18: activity_second.

Archivos de programa

```
package bancho.com.getresultssubactivity;

import androidx.appcompat.app.AppCompatActivity;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity implements View.OnClickListener{

    private static final int REQUEST_CODE = 1000; (1)
    Button b;
    TextView t;

    @Override
    protected void onCreate(Bundle savedInstanceState) {...} (2)

    public void onClick(View v) {...} (3)

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {...}
}
```

Listado 9.19: MainActivity.

(1) El código de solicitud puede ser cualquier valor; solo necesitamos marcar cada actividad que nos enviará datos.
(2) Esto contendrá todos nuestros códigos de inicialización, como es costumbre.
(3) Cuando se haga clic en el botón, se lanzará SecondActivity.

```
public void onClick(View v) {
    Intent intent = new Intent( packageContext: this, SecondActivity.class); (1)
    startActivityForResult(intent, REQUEST_CODE); (2)
}
```

Listado 9.20: Método onClick.

(1) Esto crea solo el objeto intencional explícito habitual.

(2) El método startActivity simplemente iniciará otra pantalla; estamos utilizando el método startActivityForResult porque esperamos que la actividad objetivo nos devuelva algunos datos. El código de solicitud servirá como un marcador que podemos usar más adelante cuando el resultado nos llegue, y podemos usar ese valor para enrutar la lógica del programa, en caso de que nuestra aplicación inicie varias actividades.

Pasamos a SecondActivity, donde un texto de edición espera la entrada. Cuando se hace clic en el botón, simplemente recuperamos el contenido del campo de texto y lo inyectamos a un objeto Intent.

El listado 9.21 muestra el código para eso.

```
b.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent();
        EditText e = (EditText) findViewById(R.id.editText);
        String data = e.getText().toString();
        intent.putExtra( name: "secondactivity", data); (1)
        setResult(Activity.RESULT_OK, intent); (2)
        finish(); (3)
    }
});
```

Listado 9.21: SecondActivity.

(1) Inyectamos el contenido de la variable de datos en el objeto de intención usando el método putExtra. Estamos asignando una llave de “secondactivity”, y necesitamos usar la misma clave cuando la recuperemos más tarde.

(2) Esto preparará los datos que estamos por devolver.

(3) Esto matará SecondActivity. Tan pronto como muera, el tiempo de ejecución llamará a MainActivity en ActivityResult.

Ahora volvemos a MainActivity después de que SecondActivity muere. Debido a que SecondActivity estableció el código de resultado y ha devuelto algunos datos a MainActivity, el tiempo de ejecución llama a onActivityResult. El listado 9.22 recorre el código.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    if ((requestCode == REQUEST_CODE) && (resultCode == Activity.RESULT_OK)) { (1)
        t.setText(data.getStringExtra( name: "secondactivity")); (2)
    }
}
```

Listado 9.22: Cuando el resultado retorna.

(1) Verificamos si el código de solicitud proviene de una actividad que nos interesa. Además, debemos verificar si el código de resultado es RESULT_OK, lo que significa que la operación fue exitosa. De lo contrario, es posible que debas agregar, al menos, algunos códigos de registro aquí.

(2) Recopilamos los datos con la clave “secondactivity”; esta es la misma clave que utilizamos en SecondActivity, cuando inyectamos los datos en la intención

```java
package bancho.com.getresultssubactivity;

import androidx.appcompat.app.AppCompatActivity;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity implements View.OnClickListener{

    private static final int REQUEST_CODE = 1000;
    Button b;
    TextView t;

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

        b = (Button) findViewById(R.id.button);
        t = (TextView) findViewById(R.id.textView);
        b.setOnClickListener(this);
    }

    public void onClick(View v) {
        Intent intent = new Intent(this, SecondActivity.class);
        startActivityForResult(intent, REQUEST_CODE);
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        if ((requestCode == REQUEST_CODE) && (resultCode == Activity.RESULT_OK)) {
            t.setText(data.getStringExtra("secondactivity"));
        }
    }
}
```

Listado 9.23: Código completo para MainActivity.

```
package bancho.com.getresultssubactivity;

import androidx.appcompat.app.AppCompatActivity;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class SecondActivity extends AppCompatActivity {

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

        Button b = (Button) findViewById(R.id.button2);
        b.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent();
                EditText e = (EditText) findViewById(R.id.editText);
                String data = e.getText().toString();
                intent.putExtra("secondactivity", data);
                setResult(Activity.RESULT_OK, intent);
                finish();
            }
        });
    }
}
```

Listado 9.24: Código completo para SecondActivity.

Intenciones implícitas

El enfoque de Android para la interactividad del programa es de hecho único porque está muy centrado en el usuario. Le da al usuario una gran cantidad de poder para tomar decisiones sobre cómo manipular y crear datos.

Tomemos un escenario de uso común para un dispositivo Android. Un usuario abre la aplicación “Contactos” y elige los detalles de contacto de John Doe, por ejemplo. Este contacto podría tener una dirección de correo electrónico, un teléfono móvil y un nombre de Twitter, por ejemplo.

El usuario podría tocar todos y cada uno de los puntos de contacto de John, y cada vez, Android lanzará una aplicación diferente, el cliente de correo electrónico predeterminado, un marcador y una aplicación de Twitter descargada. El usuario probablemente no le importa qué aplicación se inició o cuántas aplicaciones están actualmente abiertas; él solo quiere

enviar un mensaje. Si a este usuario no le gusta la aplicación de correo electrónico o la aplicación predeterminada de Twitter, podría eliminar estas aplicaciones y reemplazarlas por otra cosa, y él debería volver a estar en el negocio.

Para que este tipo de interacción de programa suceda, Android necesitaba diseñar la plataforma centrándose en el acoplamiento flexible y la capacidad de conexión. Un componente (como la aplicación de contactos) no debe conocer ningún detalle específico sobre qué aplicación debe usar cuando se toca una dirección de correo electrónico o un número de teléfono móvil.

La resolución de qué tipo de aplicación usar para un tipo específico de datos no debe estar integrada en la aplicación de contactos; de lo contrario, el usuario no podrá usar su elección de correo electrónico o la aplicación de Twitter.

Aquí es donde entran los intentos; la idea básica es que cuando un componente tiene datos o información que está más allá de su capacidad de servicio, puede ir a la plataforma Android, utilizando intenciones, y preguntar si hay alguna aplicación que pueda (o quiera) hacer eso.

Los intentos nos permiten utilizar las capacidades de otras aplicaciones simplemente creando y lanzando el intento, sin especificar el componente de destino: si especificas el componente de destino, se convertirá en un intento explícito.

La sintaxis general para crear intenciones implícitas es la siguiente:

```
Intent intent = new Intent();          (1)
intent.setAction(ACTION);              (2)
intent.setData(DATA);                  (3)
startActivity(intent);                 (4)
```

(1) Crear el objeto de intención.

(2) Especificar la acción. Estas acciones son constantes de la clase Intent: por ejemplo, Intent.ACTION_VIEW.

(3) Especifica los datos, si hay alguno.

(4) Inicia la actividad

Exploremos los intentos implícitos en un proyecto de demostración y veamos cómo se combinan estas cosas en el código.


1:46
GetResultsSubActivity
LAUNCH 2ND ACTIVITY


1:46
GetResultsSubActivity
Type something
CLOSE

Proyecto de demostración

Creamos un proyecto de demostración, la Tabla 9.6 muestra los detalles de este proyecto.

Puedes consultar el Listado 9.25 y la figura 9.16 para ver los detalles de la UI.

Application Name	ImplicitIntents
Project location	Deja el valor predeterminado. Ignora el soporte de C++ y Kotlin.
Form factor	Solo teléfono y tableta.
Minimum SDK	API 26 Oreo
Type of activity	Vacío (Empty)
Activity name	MainActivity (defecto)
Layout name	activity_main. (defecto)

Tabla 9.6: ImplicitIntents detalles del proyecto.

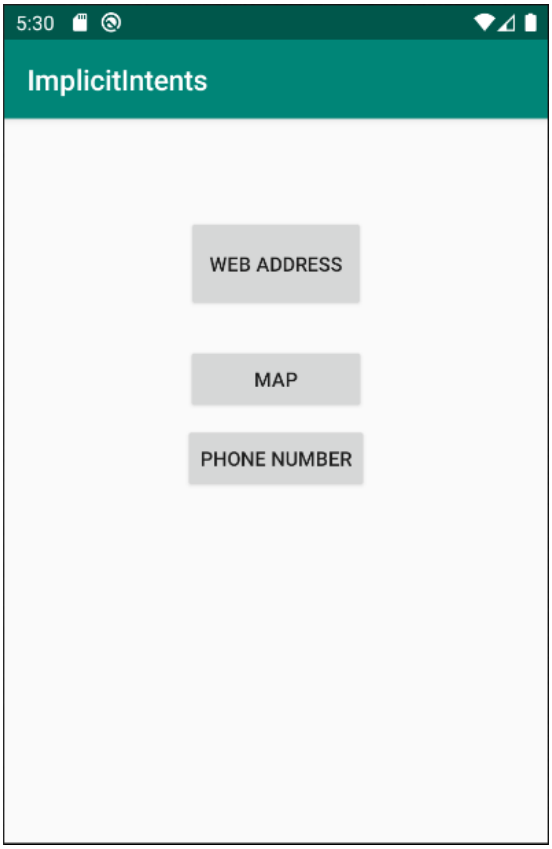

Figura 9.6: Intenciones implícitas.

```
<?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">

    <Button
        android:id="@+id/btnwebaddress"
        android:layout_width="126dp"
        android:layout_height="0dp"
        android:layout_marginBottom="80dp"
        android:layout_marginTop="69dp"
        android:text="Web Address"
        app:layout_constraintBottom_toTopOf="@id/btnphonenumber"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>

    <Button
        android:id="@+id/btnmap"
        android:layout_width="127dp"
        android:layout_height="48dp"
        android:layout_marginTop="24dp"
        android:text="Map"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@id/btnwebaddress"/>

    <Button
        android:id="@+id/btnphonenumber"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="264dp"
        android:layout_marginTop="24dp"
        android:text="Phone Number"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@id/btnmap" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Listado 9.25: activity_layout.

```
package bancho.com.implicitintents;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.util.Log;
import android.view.View;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {...}

    @Override
    public void onClick(View v) {...}
}
```

Listado 9.26: Esqueleto de MainActivity.

Esta es una estructura muy básica para el programa principal. Utilizaremos MainActivity como el objeto detector y luego implementaremos onClick como un método de miembro reemplazado. Hay tres botones en la aplicación. El botón “dirección web” intenta resolver una solicitud http, el botón “mapa” intenta resolver un código geográfico y el botón “número de teléfono”" intenta resolver un número de teléfono.

Abrir una solicitud http

Para manejar una solicitud http, como lo que nuestro botón de “dirección web” intentará hacer, tenemos que hacer un par de cosas. En primer lugar, necesitaremos un objeto URI; esto puede ser administrado por el siguiente código

Uri uri = Uri.parse ("http://inteitfcc2020.epizy.com/");

El método (parse) de análisis del objeto Uri debe poder tomar un objeto String que especifique un URL web y devolvernos un objeto URI adecuado. El siguiente paso es crear un intento. Podemos usar el constructor no-arg del Intent para hacer esto.

Intent intent = new Intent();

Después de crear el intento, ahora podemos establecer su acción usando el siguiente código.

intent.setAction (Intent.ACTION_VIEW);

Establecer la acción del intento ayuda al tiempo de ejecución de Android a elegir qué aplicación de nuestro dispositivo puede manejar mejor la solicitud; ACTION _VIEW es una de las constantes definidas en la clase Intent, y es lo que puedes usar si deseas manejar un URL web. Puedes encontrar más información sobre algunas de las acciones de intención más comunes del sitio web oficial de Android (consulta la figura 9.17).

Después de establecer la acción del Intent, ahora debemos establecer sus datos, para que sepa a dónde ir cuando se inicia el navegador.

Esto se puede hacer con el siguiente código.

```
intent.setData (uri);
```

Por último, tenemos que lanzar el objeto de intención.

```
startActivity (intent);
```

Podemos acortar el código pasando el Uri y la acción al constructor Intent.

```
Uri uri = Uri.parse ("http://inteitfcc2020.epizy.com/");
Intent intent = new Intent(Intent.ACTION_VIEW, uri);
startActivity (intent);
```

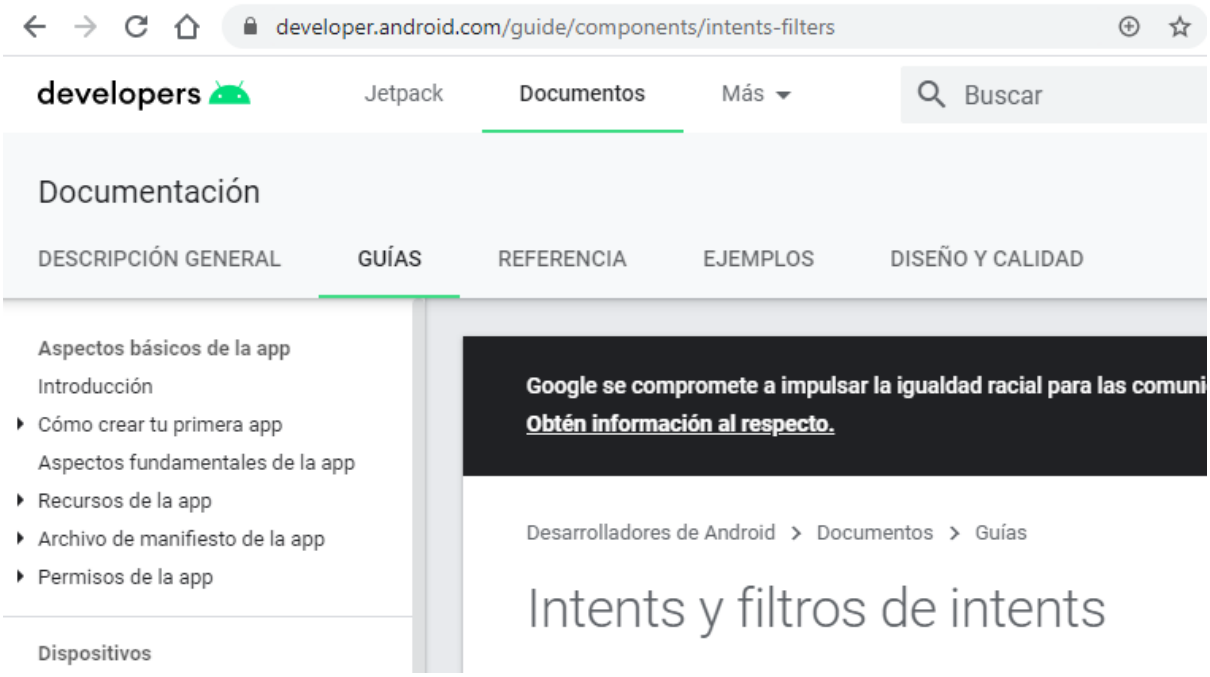

Figura 9.17: Documentación de Common Intents del sitio oficial de Android.

```java
package bancho.com.implicitintents;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.util.Log;
import android.view.View;

public class MainActivity extends AppCompatActivity implements View.OnClickListener {

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

        findViewById(R.id.btnwebaddress).setOnClickListener(this);
        findViewById(R.id.btnmap).setOnClickListener(this);
        findViewById(R.id.btnphonenumber).setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {

        Uri uri = null;
        Intent intent = null;

        switch(v.getId()) {
            case R.id.btnwebaddress:
                uri = Uri.parse("https://developer.android.com/reference/android/content/Intent");
                intent = new Intent(Intent.ACTION_VIEW, uri);
                startActivity(intent);
                break;
            case R.id.btnmap:
                uri = Uri.parse("geo:40.7113399,-74.0263469");
                intent = new Intent(Intent.ACTION_VIEW, uri);
                startActivity(intent);
                break;
            case R.id.btnphonenumber:
                uri = Uri.parse("tel:3334589076");
                intent = new Intent(Intent.ACTION_DIAL, uri);
                startActivity(intent);
                break;
                default:
                    Log.i(getClass().getName(), "No se puede resolver el clic del boton");
        }
    }
}
```

(1) (2) (3)

Listado 9.27: Código completo para MainActivity.

(1) Inicia un navegador y toma el sitio web de Android (Intents) (consulta la figura 9.18).

(2) Abre Google Maps y muestra la ubicación del código geográfico especificado (consulta la figura 9.18).

(3) Inicia la aplicación del marcador (consulta la figura 9.18).

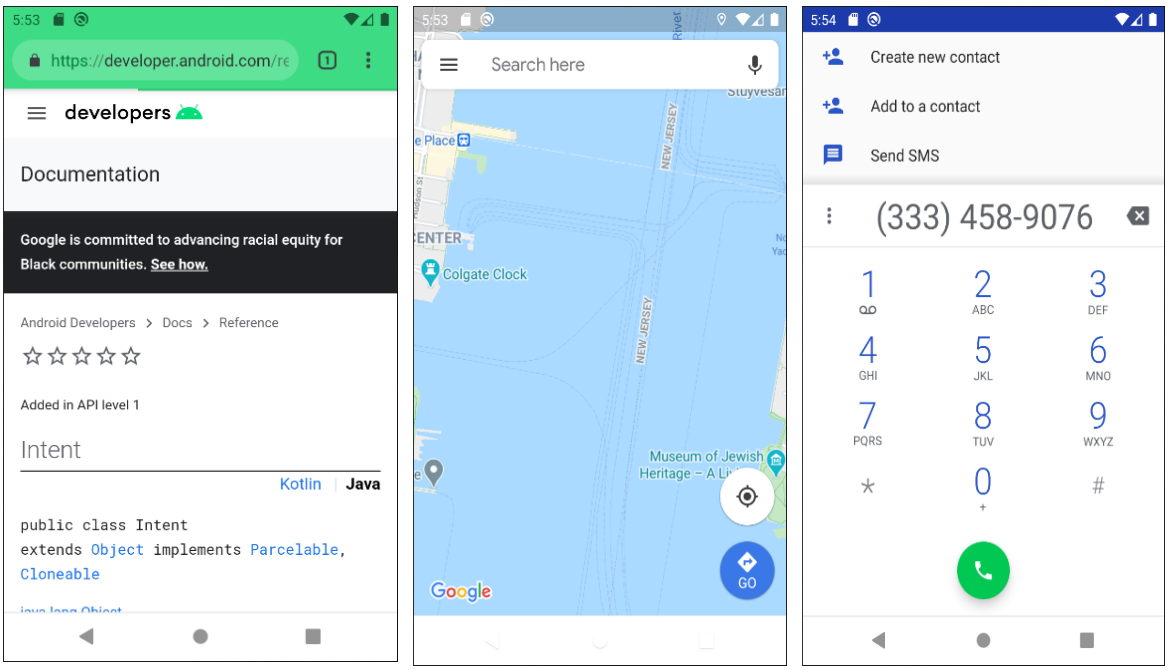

Figura 9.18: Salida de la aplicación.

Actividad del ciclo de vida

Las aplicaciones móviles no son realmente aplicaciones de escritorio que se ejecutan en una pantalla más pequeña. No utilizamos aplicaciones móviles de la misma forma que usamos aplicaciones de escritorio.

Cuando usamos una aplicación de escritorio, por lo general permanece abierta y activa durante bastante tiempo porque nos centramos en la tarea que tenemos entre manos. Las aplicaciones móviles, por otro lado, tienen una vida útil más corta.

Normalmente lo sacamos del bolsillo (al dispositivo móvil), hacemos algo rápido y luego lo devolvemos. A veces, incluso, cuando estamos usando una aplicación en particular, podemos vernos interrumpidos por otra aplicación (por ejemplo, una llamada telefónica), por lo que la aplicación original que estábamos viendo quedaría eclipsada por la aplicación de interrupción. Toda la activación y malabares de estas aplicaciones son administradas por el tiempo de ejecución de Android.

Como desarrollador, no tienes el control del ciclo de vida de la aplicación; es el usuario. No puedes suponer que el usuario no será interrumpido cuando está por ingresar datos a su aplicación. Tampoco puedes asumir que tu aplicación siempre saldrá con gracia; podría matarse sin tener la oportunidad de cerrar correctamente. Debes estar a la defensiva al diseñar

su código y considerar que estas cosas pueden suceder. Afortunadamente, el tiempo de ejecución de Android nos notifica cada vez que algo sucede con nuestros componentes (como la actividad).

Esta sección explora los diversos eventos que le pueden suceder a una actividad a lo largo de su ciclo de vida, desde el punto de creación hasta su destrucción final. La figura 9.19 muestra el ciclo de vida de la actividad.

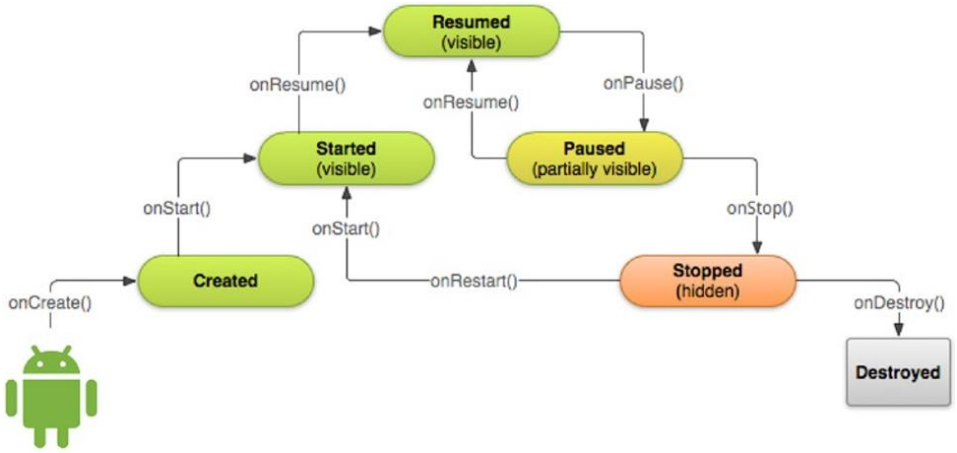

Figura 9.19: Ciclo de vida de la Actividad.

Event	Descripción
onCreate	Se llama cuando se crea por primera vez la actividad; puedes poner tus códigos de inicialización aquí.
onRestart	Cuando la actividad se detuvo y se reinició nuevamente. Esto siempre es seguido por onStart.
onStart	Cuando la actividad comienza a ser visible para el usuario.
onResume	La actividad está lista para interactuar con el usuario, en este punto; la actividad está en la parte superior de la pila de actividades y ocupa toda la pantalla.
onPause	Cuando la actividad está a punto de pasar al segundo plano; esto puede suceder cuando otra actividad atrapa el foco .
onStop	Cuando la actividad ya no es visible para el usuario.
onDestroy	Llamado cuando la actividad es destruida. Para que la aplicación vuelva, debe crearse nuevamente.

Tabla 9.7: Métodos Callback.

El listado 9.28 muestra los métodos del ciclo de vida anulados para una actividad.

```java
package bancho.com.activitylifecycle;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.util.Log;

public class MainActivity extends AppCompatActivity {

    private String TAG;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TAG = getClass().getSimpleName();
        Log.i(TAG, "onCreate");
    }

    @Override
    protected void onStart() {
        super.onStart();
        Log.i(TAG, "onStart");
    }

    @Override
    protected void onResume() {
        super.onResume();
        Log.i(TAG, "onResume");
    }

    @Override
    protected  void onPause() {
        super.onPause();
        Log.i(TAG, "onPause");
    }

    @Override
    protected void onStop() {
        super.onStop();
        Log.i(TAG, "onStop");
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        Log.i(TAG, "onDestroy");
    }
}
```

Listado 9.28: MainActivity, Métodos del Ciclo de Vida.

Puedes ver los mensajes de registro en la ventana de la herramienta Logcat.

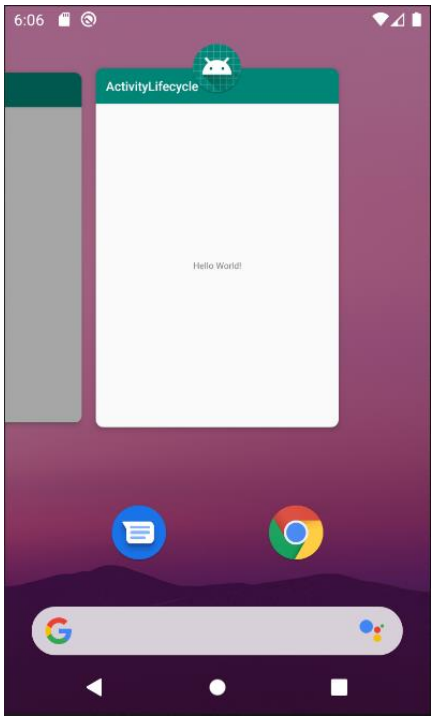

Figura 9.20: Botón de aplicaciones recientes.

Tip. Si deseas ver el mensaje onDestroy, puedes usar el botón “Aplicaciones recientes” del dispositivo (si tienes uno) para ver todas las aplicaciones en ejecución (Figura 6-20). Puedes matar la aplicación desde allí.