Navegación

Qué cubriremos en esta práctica:

- Revisión de técnicas de navegación en Android
- Componentes de navegación de Jetpack

Navegación antes que los componentes de la arquitectura

En los primeros días del desarrollo con Android, la mayoría de las aplicaciones no triviales tenían más de una pantalla y la interfaz de usuario estaba dividida en múltiples actividades. Eso significaba que necesitabas la habilidad para navegar de una actividad a otra y viceversa. Entonces, durante esos días, es posible que hayas escrito algo que se parezca al código del listado 10.1.

```
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 { }
```

Listado 10.1: Cómo iniciar una actividad.

Si necesitabas pasar datos de una actividad a otra, es posible que los hayas codificado como el fragmento de código que se muestra en el listado 10.2.

```
Intent intent = new Intent(this, SecondActivity.class);
Intent.putExtra("key", value);
startActivity(intent);
```

Listado 10.2: Cómo pasar datos a otra actividad.

Este tipo de gestión de pantalla tiene las siguientes ventajas:

- Es simple de hacer; simplemente llamas al método **startActivity()** desde cualquier actividad.

- La actividad que se está ejecutando actualmente se puede cerrar mediante programación llamando al método **finish()**. El usuario también puede cerrar la Activity presionando el botón Atrás.
- La pila de actividades está completamente administrada por Android Runtime; ver la Figura 10.1.

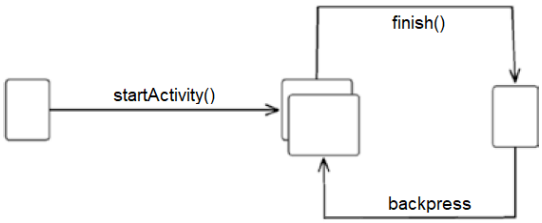

Figura 10.1: Flujo de trabajo (workflow) de una actividad simple.

Pero no todo está bien; la navegación de actividades viene con algo de equipaje. Las desventajas son:

- No tenemos una idea clara de qué Actividades están en la pila porque no las administramos; aquí hay un ejemplo de un rasgo que es tanto una ventaja como una desventaja al mismo tiempo.
- Cada pantalla requiere una nueva actividad, que agota los recursos informáticos.
- Cada actividad se declaró en el archivo de manifiesto de Android, lo que Android Studio hace automáticamente cada vez que creas una actividad con los asistentes, por lo que no es un gran problema; era un problema antes de que apareciera Android Studio.
- Te resultará difícil utilizar los patrones de navegación más modernos, como la barra de navegación inferior.

Debido a estas limitaciones, surgió otra forma de navegación por la pantalla. En algún momento de 2011, cuando Google lanzó Android 4.0, obtuvimos **Fragments**. Acabamos de tratar Fragmentos en la práctica anterior, así que estoy seguro de que todavía está muy nuevo para ti; una actividad es básicamente una unidad de composición para la interfaz de usuario, y el fragmento es solo una unidad de composición más pequeña.

Un fragmento, como una actividad, se compone de dos partes: un programa Java o Kotlin y un archivo de diseño. La idea es básicamente la misma: define la interfaz de usuario en un archivo XML y luego infla el archivo XML durante el tiempo de ejecución para que toda la interfaz de usuario del archivo XML se convierta en objetos Java reales.

La idea es crear múltiples fragmentos y contenerlos en una sola Actividad. Por lo general, ocultarías o mostrarías un Fragmento dependiendo de la acción del usuario, la orientación

del dispositivo o el factor de forma del dispositivo; y esto generalmente se hace con los objetos FragmentManager y FragmentTransaction. El fragmento de código del listado 10.3 es un código de gestión de fragmentos típico.

```
FragmentManager fm = getFragmentManager();
FragmentTransaction ft = fm.beginTransaction();
Fragment fragment = new FirstFragment();
ft.add(R.id.fragment_container, fragment);
ft.commit();
```

Listado 10.3: Fragmento de un fragmento.

Con Fragments, sabemos exactamente qué hay en la pila de navegación, a diferencia de cuando usamos la navegación por actividad; pero, como puedes ver en el listado 10.3, puede resultar engorroso porque debemos administrar manualmente la pila de navegación.

Hasta ahora, solo teníamos dos opciones para la navegación: o usamos la navegación basada en actividades, que era fácil y simple de usar, pero tenemos una penalización en el rendimiento y no tenemos control sobre la pila de navegación, o usamos Fragmentos que ofrecían usar el control total de la pila de navegación, pero la API es engorrosa y propensa a errores.

Avancemos rápido hasta 2017 cuando Google presentó los componentes de navegación. Ahora podemos usar Fragments pero sin el equipaje de la complicada API. Con los componentes de navegación, todos los códigos que hemos escrito en el listado 10.3 ahora se pueden reemplazar con una sola línea de código (consulta el listado 10.4).

```
findNavController().navigate(destination);

// WE NO LONGER NEED THE FF CODES
/*
  FragmentManager fm = getFragmentManager();
  FragmentTransaction ft = fm.beginTransaction();
  Fragment fragment = new FirstFragment();
  ft.add(R.id.fragment_container, fragment);
  ft.commit();
*/
```

Listado 10.4: Fragmento de componente de navegación.

Componentes de navegación

Muy bien, esa única línea de referencia de código de la sección anterior probablemente te emocionó y alivió. Pero no es el ahorro de pulsaciones de teclas el panorama general aquí, es el hecho de que ahora podemos obtener lo mejor de la navegación basada en actividades y en fragmentos. Ahora, la navegación por fragmentos también tiene una API sencilla.

Pero primero, debemos comprender un poco los componentes de navegación. Es una pequeña parte de los componentes de Arquitectura, que a su vez es parte de una cosa más grande llamada Android Jetpack. No vamos a entrar en Jetpack ni en los componentes de Arquitectura en detalle aquí; son temas extensos, pero un breve trasfondo no está de más.

En Google I/O 2017, Google presenta los componentes de Arquitectura de Android. Estas bibliotecas son parte de una colección más grande llamada Android Jetpack. Junto con los componentes de Arquitectura, había otros como Foundation, Behavior y UI.

Jetpack es una colección de componentes de software de Android. Nos ayuda a seguir las mejores prácticas y nos permite evitar escribir demasiado código repetitivo. Encontrarás los códigos de Jetpack en las bibliotecas de paquetes de androidx.*.

Aquí hay una breve descripción de los componentes de Jetpack:

Foundation

- **AppCompat**: Te permite escribir código que se degrada con facilidad en versiones anteriores de Android
- **Android KTX**: Escribe un código Kotlin más conciso e idiomático, si estás usando Kotlin.
- **Multidex**: Brinda soporte para aplicaciones con múltiples archivos DEX
- **Test**: Un framework de prueba para pruebas de IU unitarias y en tiempo de ejecución

Behavior

- **Download manager**: Te permite escribir programas que programen y administren grandes descargas.
- **Media and playback**: API compatibles con versiones anteriores para la reproducción y el enrutamiento de medios.
- **Notifications**: Proporciona una API de notificación compatible con versiones anteriores con soporte para wear y auto.

- **Permissions**: API de compatibilidad para comprobar y solicitar permisos de aplicaciones.
- **Preferences**: Crea pantallas de configuración interactivas.
- **Sharing**: Proporciona una acción para compartir adecuada para la barra de acciones de una aplicación.
- **Slices**: Crea elementos de interfaz de usuario flexibles que puedan mostrar datos de la aplicación fuera de la aplicación.

UI

- **Animations y transitions**: Mueve widgets y transiciones entre pantallas.
- **Auto**: Si estás trabajando en aplicaciones que se ejecutarán en las consolas de información y entretenimiento de los vehículos, lo necesitarás. Estos son los componentes que te ayudarán a crear aplicaciones para Android Auto.
- **Emoji**: Habilita una fuente de emoji actualizada en plataformas más antiguas.
- **Fragment**: Todos los códigos de Fragmento ya se movieron aquí.
- **Layout**: Diseño de widgets utilizando diferentes algoritmos.
- **Palette**: Extrae información útil de las paletas de colores.
- **TV**: Componentes para ayudar a desarrollar aplicaciones para Android TV.
- **Wear OS by Google**: Si deseas trabajar con dispositivos portátiles de Android como el reloj, esto es lo que necesitas.

Architecture

- **Data binding**: Enlaza de forma declarativa datos observables a elementos de la interfaz de usuario.
- **Lifecycles**: Gestiona los ciclos de vida de las actividades y los fragmentos.
- **LiveData**: Notifica a las vistas cuando cambia la base de datos subyacente.
- **Paging**: Carga gradualmente información a pedido desde tu fuente de datos; Piensa que cuando el usuario se desplaza por una lista, esto le ayuda a manejar la carga de datos. Se combina con la vista Recycler.
- **ViewModel**: Administra los datos relacionados con la interfaz de usuario de una manera consciente del ciclo de vida.
- **WorkManager**: Gestiona trabajos en segundo plano.
- **Navigation**: Implementación de la navegación en una aplicación. Pasa datos entre pantallas. Proporciona enlaces profundos desde fuera de la aplicación.
- **Room**: Piensa en ORM para tu base de datos SQLite.

Hay mucho que explorar en Jetpack, así que asegúrate de echarles un vistazo.

Volviendo a nuestro tema, los componentes de navegación simplifican la implementación de, bueno, la navegación entre destinos en una aplicación. Un destino es cualquier lugar de tu aplicación. Puede ser una actividad, un fragmento dentro de una actividad o una vista personalizada; y los destinos se gestionan mediante un grafo de navegación.

Un grafo de navegación agrupa todos los destinos y define las diferentes conexiones entre los destinos; estas conexiones se llaman acciones (**actions**). El grafo es simplemente un archivo de recursos XML que representa todas las rutas de navegación de tu aplicación. Puedes tener más de un grafo de navegación en tu aplicación.

Trabajar con Jetpack Navigation

Para apreciar mejor los componentes de navegación, es mejor si puedes trabajar en un proyecto pequeño. Entonces, crea un nuevo proyecto vacío en Android Studio, como se muestra en la figura 10.2.

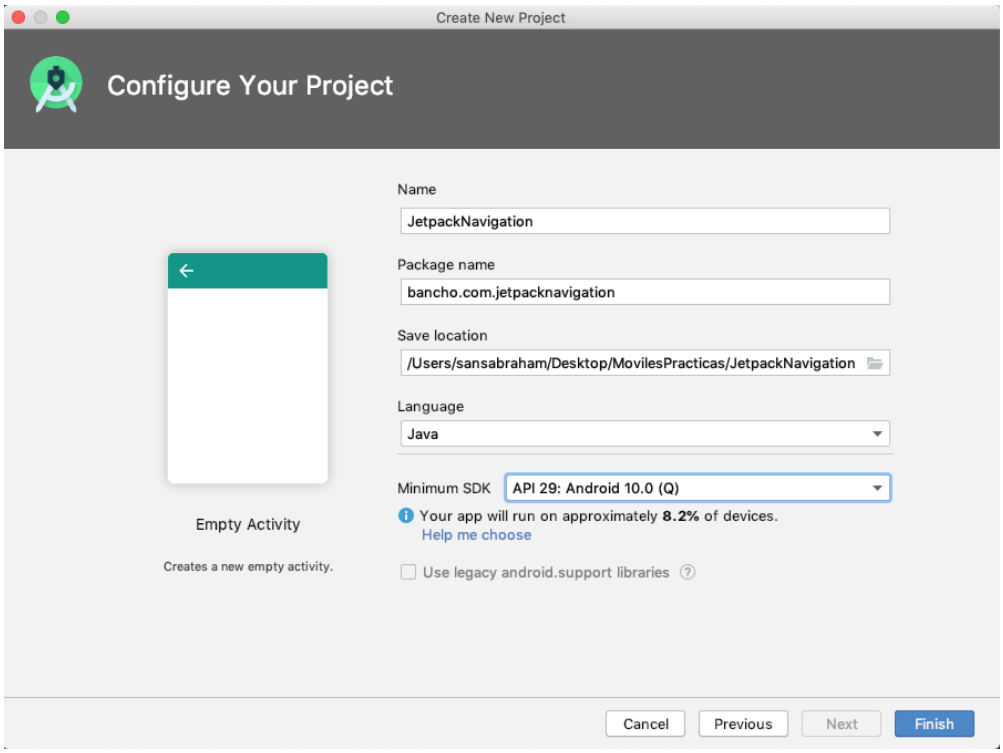

Figura 10.2: Nuevo proyecto vacío.

A continuación, debemos agregar las dependencias del componente de navegación. Puedes hacer esto en el archivo **build.gradle** de nivel de módulo (que se muestra en la figura 10.3).

Ten en cuenta que hay dos construcciones. archivos gradle; quieres el que dice (Module:app). Sólo hay darle clic al archivo seleccionado.

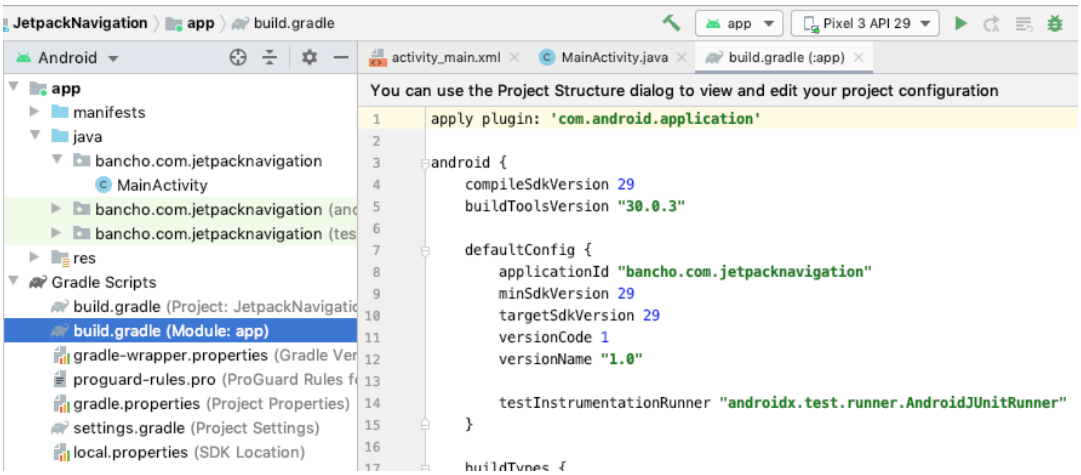

Figura 10.3: Archivo build.gradle (nivel de módulo).

Edita este archivo para que coincida con el código del listado 10.5.

```
dependencies {
    def nav_version = "2.2.2"
    implementation "androidx.navigation:navigation-fragment:$nav_version"
    implementation "androidx.navigation:navigation-ui:$nav_version"

    implementation fileTree(dir: "libs", include: ["*.jar"])
    implementation 'androidx.appcompat:appcompat:1.3.1'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.0'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'

}
```

Listado 10.5: Agregar navegación a build.gradle.

En el momento de redactar esta práctica, la versión estable de los componentes de navegación es la 2.2.2; También hubo una versión beta **2.3.0-beta01**. Para cuando leas esto, es posible que la versión se haya actualizado.

Asegúrate de consultar el sitio web oficial de desarrolladores de Android para obtener más información actualizada:
https://developer.android.com/jetpack/androidx/releases/navigation.

A continuación, agreguemos un grafo de navegación al proyecto. Puedes crear un grafo de navegación creando un nuevo archivo de recursos; haz clic con el botón derecho en la carpeta res del proyecto, luego selecciona **New ➤ Android Resource File**, como se muestra en la figura 10.4.

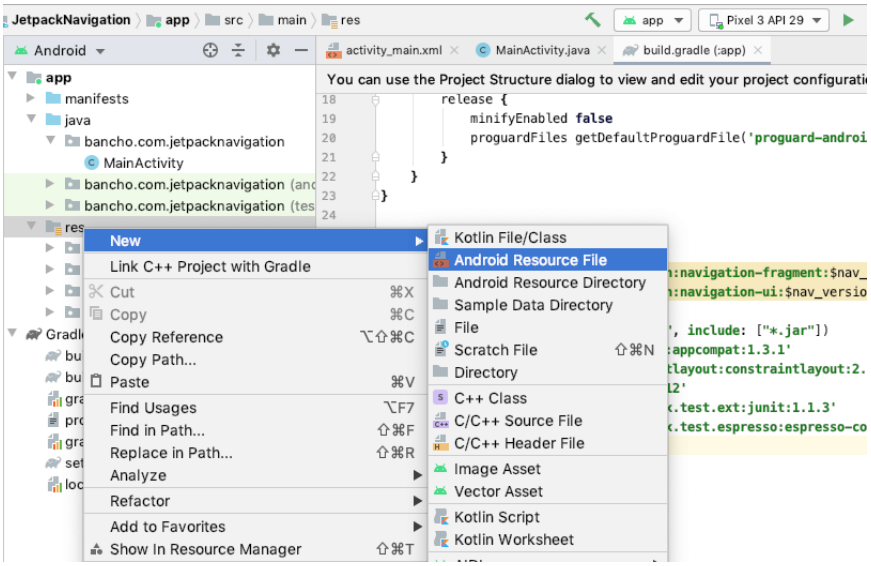

Figura 10.4: Agregar un nuevo archivo de recursos.

En el cuadro de diálogo "New Resource File", cambia el tipo de recurso a Navigation y proporciona el nombre del archivo:

- **File name**: nav_graph
- **Resource type**: Navigation (debes hacer clic en la flecha hacia abajo para seleccionarlo)

En la ventana que sigue (que se muestra en la figura 10.5), haz clic en OK.

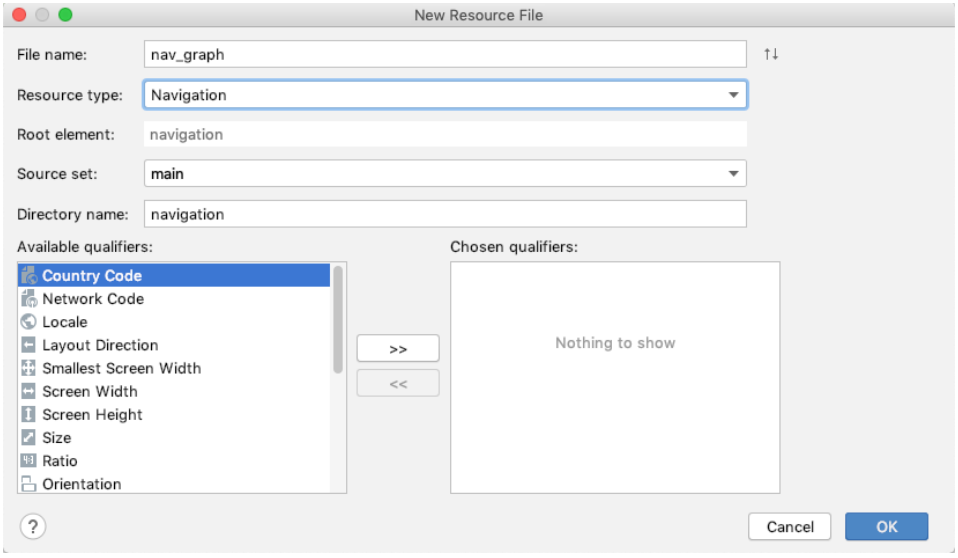

Figura 10.5: Nuevo archivo de recursos.

Cuando se crea el recurso, verás una nueva carpeta (navigation) y un nuevo archivo (nav_graph.xml) debajo de la carpeta res del proyecto, como se muestra en la figura 10.6. Android Studio abrirá el grafo de navegación recién creado en el editor. La figura 10.6 muestra el grafo de navegación recién creado; está vacío, por supuesto.

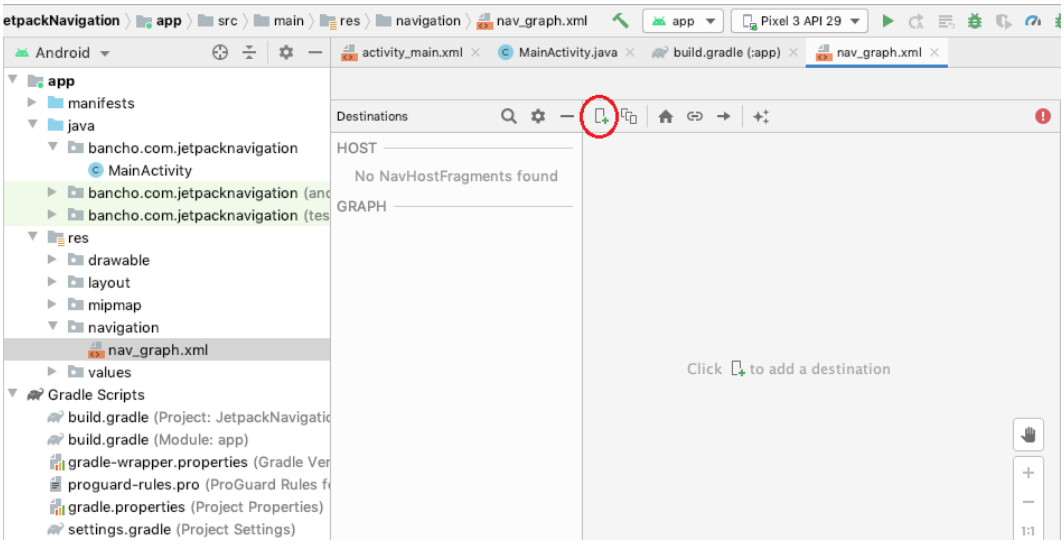

Figura 10.6: Grafo de navegación.

Cuando utilizas componentes de navegación, la navegación ocurre como una interacción entre destinos. Los destinos son los lugares a los que pueden navegar tus usuarios y los destinos se conectan mediante acciones. Por el momento, todavía no tenemos ningún destino; entonces, agreguemos uno. Haz clic en el signo más en el panel superior del editor de navegación, como se muestra en la figura 10.7, luego elije “Create new destination”.

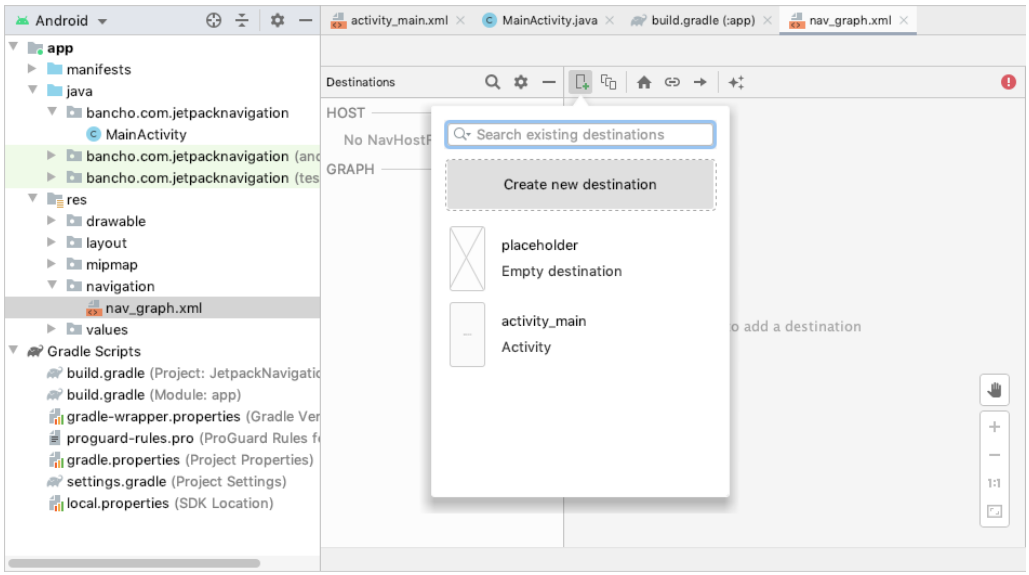

Figura 10.7: Crea un nuevo destino.

Esto mostrará el diálogo para crear un nuevo Fragmento. En la ventana que sigue (figura 10.8), elije un Fragmento en blanco.

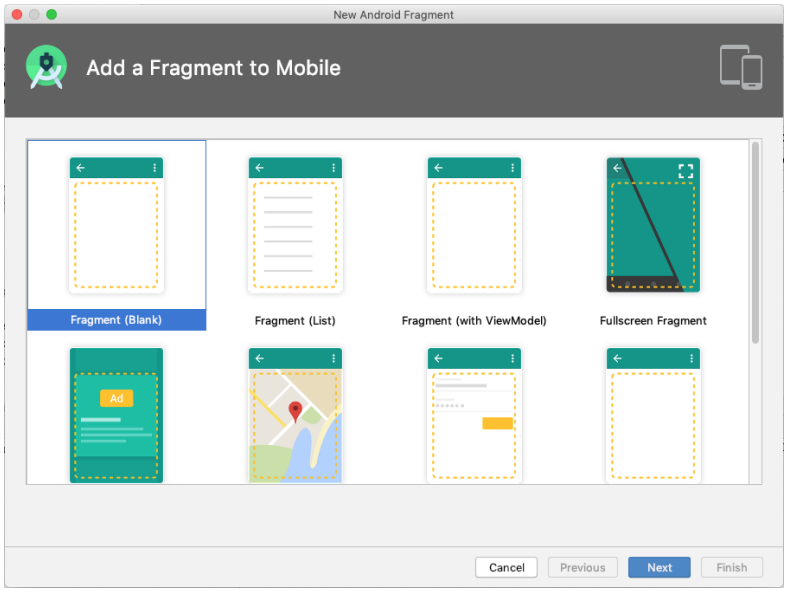

Figura 10.8: Nuevo fragmento de Android.

En la ventana que sigue, escribe el nombre del Fragmento y el nombre del archivo de diseño y elije el lenguaje de origen, como se muestra en la figura 10.9.

- **Fragment Name**: one
- **Fragment Layout Name**: fragment_one
- Mantén el lenguaje de origen como Java

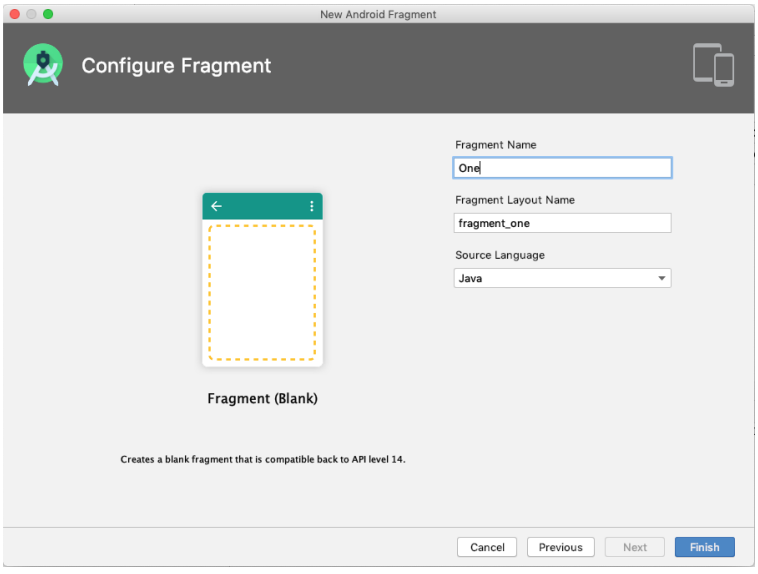

Figura 10.9: Nuevo fragmento de Android, detalles.

Haz clic en **Finish** para iniciar la creación del nuevo Fragmento; este Fragmento se convertirá en uno de los destinos de tu aplicación. Crea otro destino y haz que el nombre del fragmento sea “Two”. El editor de navegación, a estas alturas, debería parecerse a la figura 10.10.

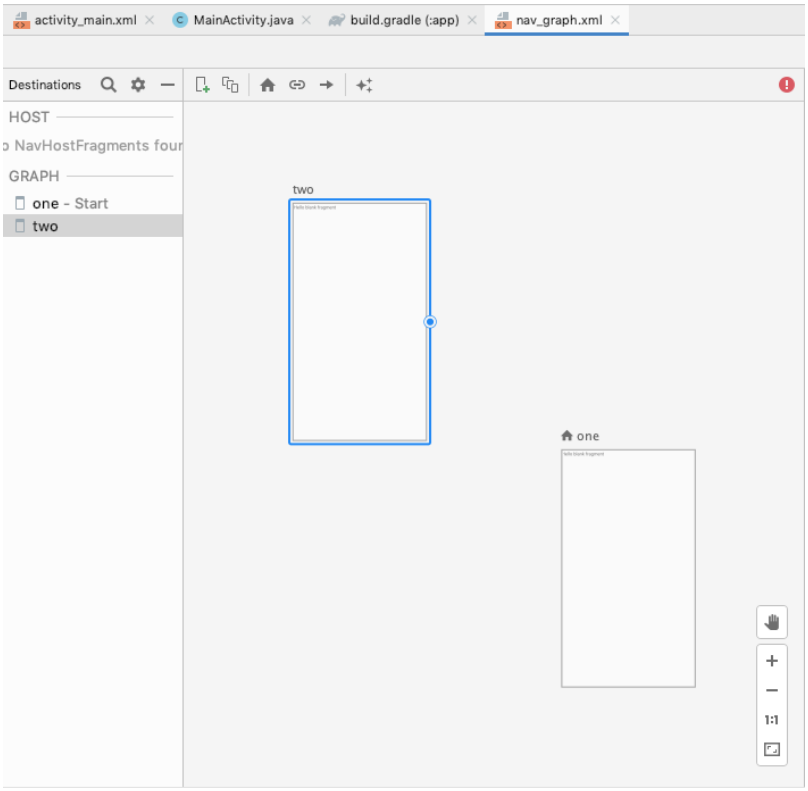

Figura 10.10: Editor de navegación.

Observa también que hay dos nuevas clases de Java (One.java y Two.java) y dos nuevos archivos de diseño (fragment_one.xml y fragment_two.xml). Estos archivos se generaron cuando creamos los destinos One y Two. Observa en la figura 10.10 que el fragmento uno tiene el ícono de inicio al lado. Esto es solo porque lo creamos primero. El destino inicial o el destino inicial es la primera pantalla que verán los usuarios. Puedes cambiar el destino de inicio en cualquier momento haciendo clic con el botón derecho en cualquier destino y luego haciendo clic en “Establecer como destino de inicio (Set as start destination)”; pero por ahora, mantendremos one como destino de inicio.

Nuestro grafo de navegación aún no tiene **NavHost**; necesita uno. Un NavHost actúa como una ventana para todos nuestros destinos. Es un contenedor vacío donde los destinos se intercambian hacia adentro y hacia afuera a medida que el usuario navega por la aplicación. NavHost debe estar en una actividad. Vamos a poner NavHost en nuestra MainActivity.

Abre el archivo de diseño para nuestra MainActivity, está en res/layout/activity_main.xml, luego edítalo en modo texto. El activity_main predeterminado contiene un único objeto

TextView; elimínalo y reemplázalo con el fragmento de código que se muestra en el listado 10.6.

```
<?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">

    <fragment
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/nav_container" (1)
        android:name="androidx.navigation.fragment.NavHostFragment" (2)
        app:navGraph="@navigation/nav_graph" (3)
        app:defaultNavHost="true" (4)
        >
    </fragment>

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

Listado 10.6: Definición de un NavHost en activity_main.xml.

(1) Necesita una identificación, como cualquier otro elemento en el archivo de recursos. Solo elegimos nav_container para este. Puedes nombrarlo como quieras.
(2) Este es el nombre completo de la clase NavHostFragment. Esta pertenece al componente de Navegación, y esta será responsable de hacer de nuestra MainActivity la ventana gráfica para todos nuestros destinos definidos.
(3) El atributo app:navGraph le dice al tiempo de ejecución qué grafo de navegación queremos alojar en MainActivity. Recuerda que puedes tener más de un grafo de navegación en la aplicación; nav_graph es el nombre que le dimos al recurso XML del grafo de navegación anteriormente.
(4) Cuando estableces defaultNavHost en verdadero, esto asegura que NavHostFragment intercepta el botón de retroceso del sistema; de esa manera, cuando el usuario hace clic en el botón Atrás, Android le mostrará la pantalla anterior en tu aplicación, y no la pantalla de una aplicación externa que estaba en la pila de actividades.

Ahora es el momento de conectar nuestros dos destinos. Abre el grafo de navegación nuevamente, está en res/navigation/nav_graph.

Queremos que el usuario navegue desde el destino uno al destino dos. Por lo tanto, coloca el mouse sobre el destino uno hasta que aparezca un pequeño círculo en su lado derecho. Haz clic y arrastra este punto hasta el destino dos, de modo que los dos destinos se puedan conectar, como se muestra en la figura 10.11.

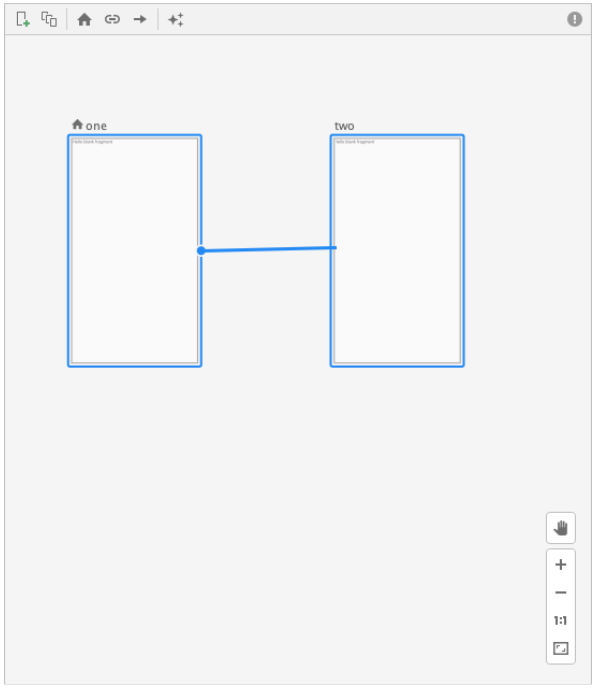

Figura 10.11: Conecte one a two.

El destino uno ahora está conectado al destino dos. Si seleccionas la conexión entre uno y dos, verás que tiene atributos que puedes establecer, como se muestra en la figura 10.2. No nos ocuparemos de los atributos; solo queremos conectar los dos destinos.

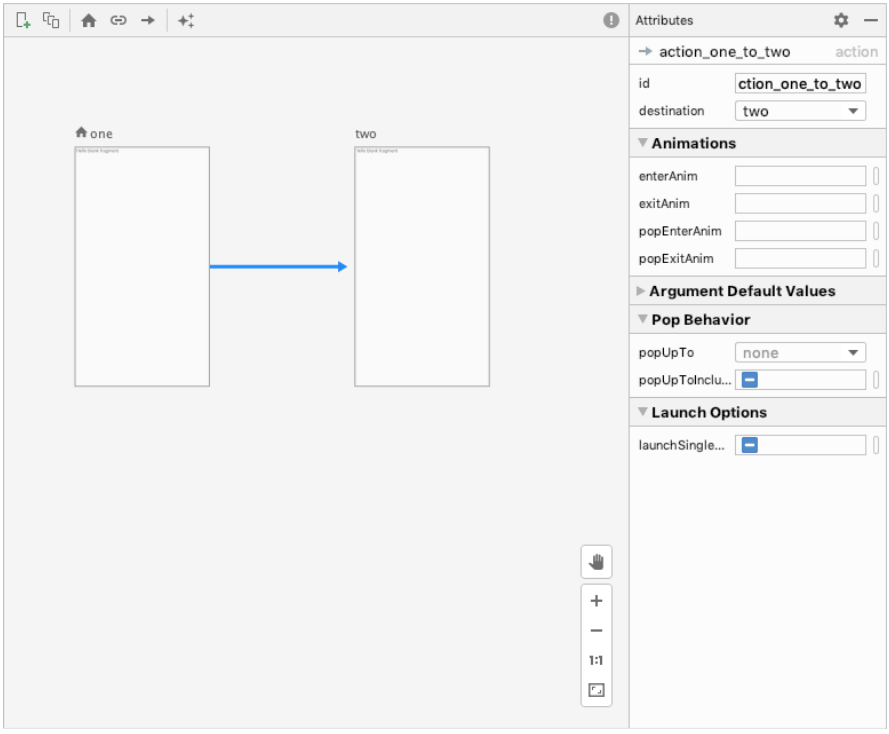

Figura 10.12: Grafo de navegación.

Para probar esta pequeña aplicación, necesitábamos un objeto en el destino de inicio que desencadenara una acción, como un Botón (que se muestra en la figura 10.13). Modifica el diseño de los dos fragmentos de la siguiente manera:

fragment_one

- Se cambió el diseño a ConstraintLayout, solo porque es más fácil trabajar con este tipo de diseño. Utiliza el diseño que sea adecuado para ti.
- Quitamos el TextView y lo reemplazamos con un botón y lo centramos.

fragment_two

- Como fragment_one, también cambiamos el diseño a ConstraintLayout.
- Cambiamos el texto del TextView y lo centramos.

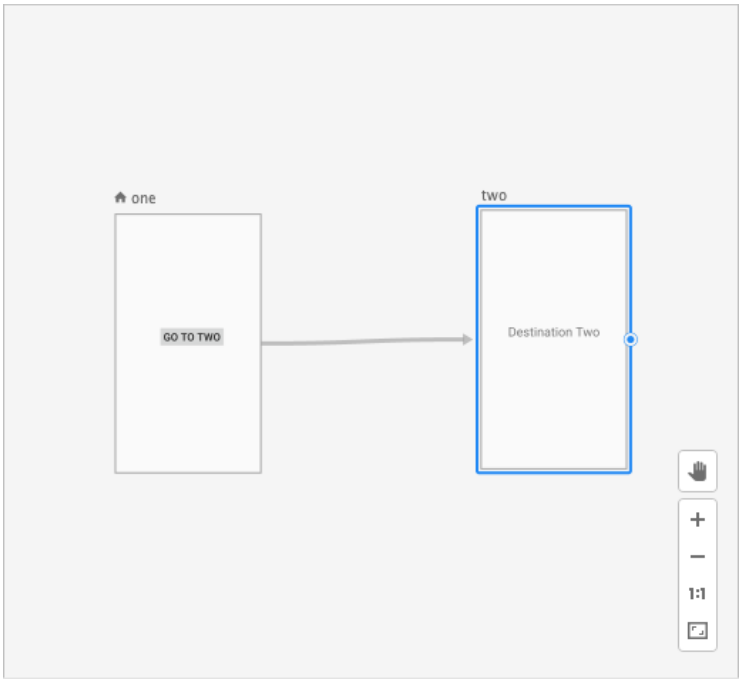

Figura 10-13. Fragmentos modificados.

A continuación, agregaremos un controlador de clics a nuestro botón, luego agregaremos el código que hará que fragment_one navegue a fragment_two cuando se haga clic en el botón.

La navegación a un destino se realiza mediante un **NavController**; es un objeto que administra la navegación de la aplicación dentro de un NavHost. Cada NavHost tiene su propio NavController correspondiente.

Un NavController te permite navegar a destinos de dos maneras: (1) navegar a un destino usando un ID, que es lo que usaremos aquí, y (2) navegar usando un URI, que dejaremos que tú lo explores.

Para agregar un controlador de clic a nuestro botón, abre One.java, que contiene el archivo fuente Java para nuestro destino One, y modifica su método onCreateView() para que coincida con el listado 10.7.

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    // Inflate the layout for this fragment

    final View view = inflater.inflate(R.layout.fragment_one, container,  attachToRoot: false);
    view.findViewById(R.id.button2).setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Navigation.findNavController(view).navigate(R.id.action_one_to_two);
        }
    });
    return view;
}
```

Listado 10.7: Clase One.onCreateView().

La línea de código más importante del listado 10.7 es el método navigate() del objeto **NavController**. Simplemente pasamos el ID de la acción que creamos en el grafo de navegación como parámetro para **navigate()**, y eso ya funcionó. Ahora puedes iniciar el emulador y probar la aplicación.

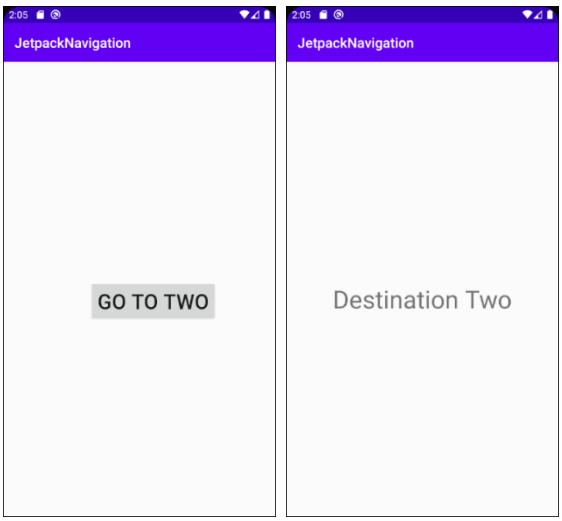

Figura 10.14: La app desarrollada.

Esta práctica simplemente arañó la superficie de los componentes de navegación; hay mucho por descubrir en esta área, así que asegúrate de echarles un vistazo.

A continuación, se anexan los códigos finales, de one.java y two.java, respectivamente.

```java
import android.os.Bundle;

import androidx.fragment.app.Fragment;
import androidx.navigation.Navigation;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

/**
 * A simple {@link Fragment} subclass.
 * Use the {@link One#newInstance} factory method to
 * create an instance of this fragment.
 */
public class One extends Fragment {

    public One() {
        // Required empty public constructor
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment

        final View view = inflater.inflate(R.layout.fragment_one, container, attachToRoot: false);
        view.findViewById(R.id.button2).setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Navigation.findNavController(view).navigate(R.id.action_one_to_two);
            }
        });
        return view;
    }
}
```

```java
package bancho.com.jetpacknavigation;

import android.os.Bundle;

import androidx.fragment.app.Fragment;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

/**
 * A simple {@link Fragment} subclass.
 * Use the {@link Two#newInstance} factory method to
 * create an instance of this fragment.
 */
public class Two extends Fragment {

    public Two() {
        // Required empty public constructor
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_two, container,  attachToRoot: false);
    }
}
```

Resumen

- Puedes seguir utilizando la navegación basada en actividades o en fragmentos en tu aplicación; solo recuerda sus pros y contras.
- Los componentes de navegación combinan las mejores características de la navegación basada en actividades y basada en fragmentos; es fácil trabajar con la API y tenemos más control sobre la pila de actividades.
- Los componentes de navegación introducen el concepto de destinos.
- Los destinos pueden ser fragmentos, actividades o vistas personalizadas; son los lugares a los que navegarán los usuarios.
- Los destinos se agrupan mediante un grafo de navegación; es un archivo de recursos XML que contiene todas las acciones entre destinos.
- Los destinos están conectados entre sí mediante acciones.
- Las ideas básicas para la navegación son:
  1. Crear un grafo de navegación.
  2. Crear destinos.

3. Conectar los destinos; cada conexión se convierte en una acción.
4. Navegar mediante programación de un destino a otro utilizando el objeto NavController. Puedes navegar utilizando un ID o un URI.