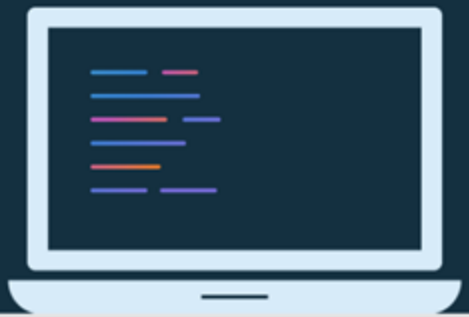
Lesson 6: App navigation



Multiple activities and intents

Multiple screens in an app

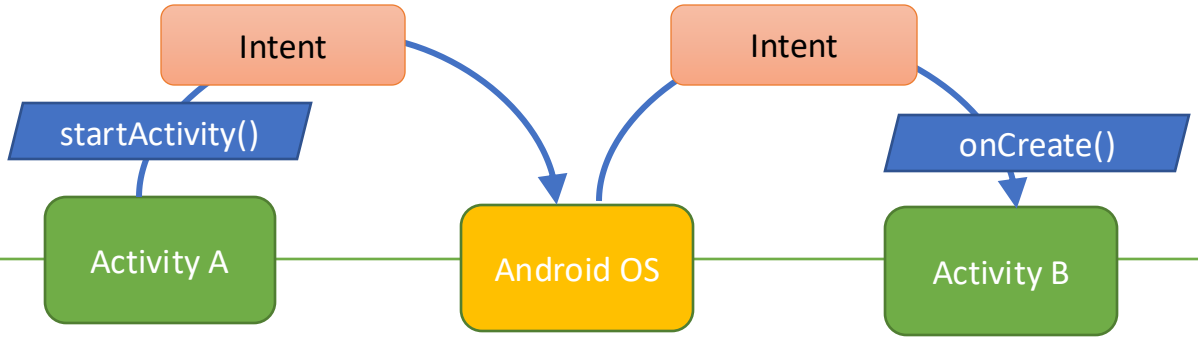
Sometimes app functionality may be separated into multiple screens.

Examples:

- View details of a single item (for example, product in a shopping app)
- Create a new item (for example, new email)
- Show settings for an app
- Access services in other apps (for example, photo gallery or browse documents)

Intents

- **Intent**: a message used by a component to request action from another app or component
- 3 main use cases for Intents
- **Case 1** (Activity A starts Activity B, no result back):
  - Call *startActivity()*, pass an Intent
  - Intent has information about Activity to start, plus any necessary data



Intents

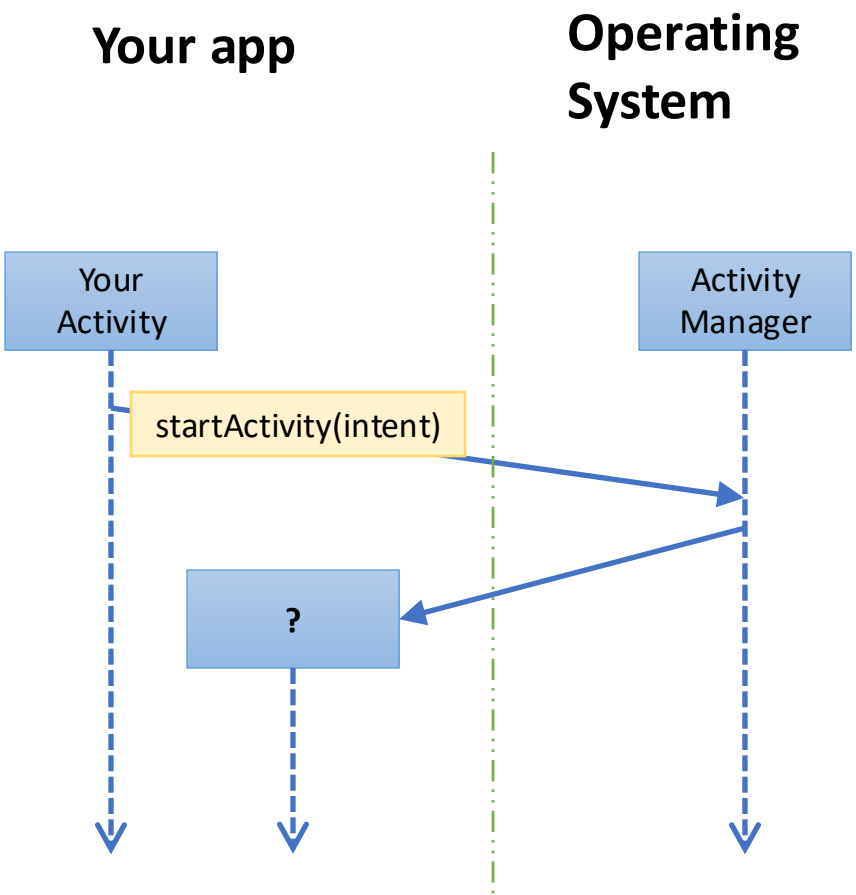
- **Case 2** (Activity A starts Activity B, gets result back):
  - Call *startActivityForResult()*, pass an Intent
  - Separate Intent received in Activity A's *onActivityResult()* callback

Intents

- **Case 3** (Activity A starts a Service):
  - E.g. Activity A starts service to download big file in the background
  - Activity A calls *startService()*, passes an Intent
  - Intent contains information about Service to start, plus any necessary data

Starting another Activity

- Activity 1 starts Activity 2
  - through the Android OS
  - by calling *startActivity(Intent)*
- Passes *Intent* (object for communicating with Android OS)
- Intent specifies which (target) Activity the *ActivityManager* should start

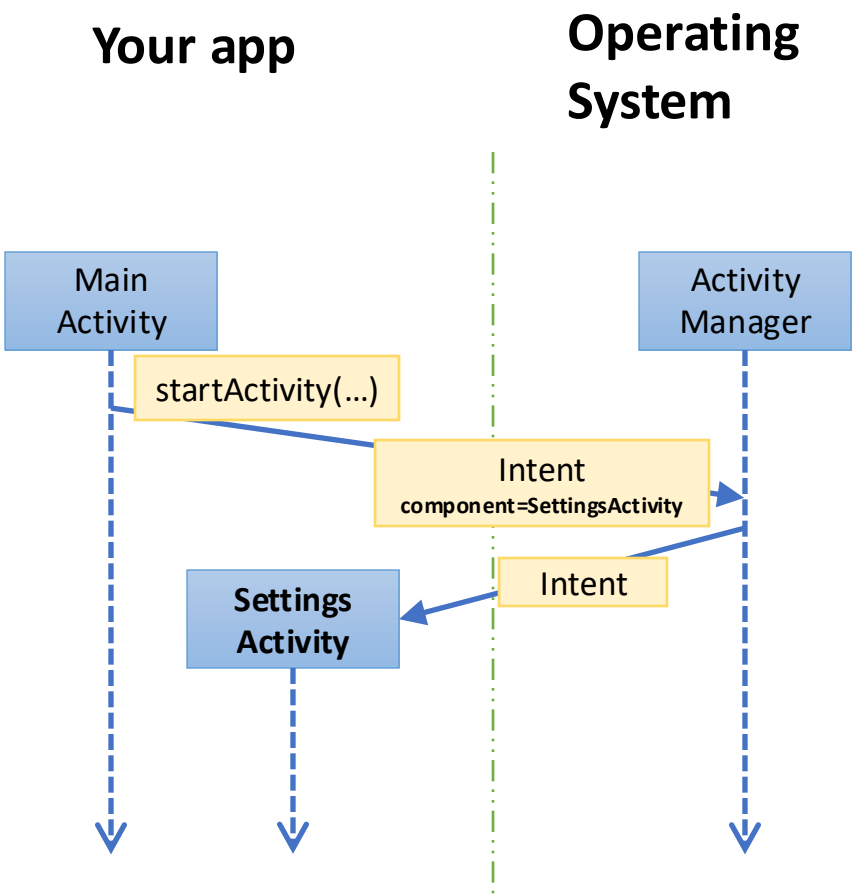


Starting another activity

- Intents have many different constructors. Example:

```
Intent(ctx: Context, cls: Class<*>)
```

- *Context*: the environment for the Intent, the starting activity
- *Class*: the activity to be started



Intent

Requests an action from another app component, such as another Activity

- An `Intent` usually has two primary pieces of information:
  - Action to be performed (for example, `ACTION_VIEW`, `ACTION_EDIT`, `ACTION_MAIN`)
  - Data to operate on (for example, a person's record in the contacts database)
- Commonly used to specify a request to transition to another Activity

Explicit intent

- Fulfills a request **using a specific component**
- Navigates internally to an Activity in your app
- Navigates to a specific third-party app or another app you've written

Explicit intent examples

Navigate between activities in your app:

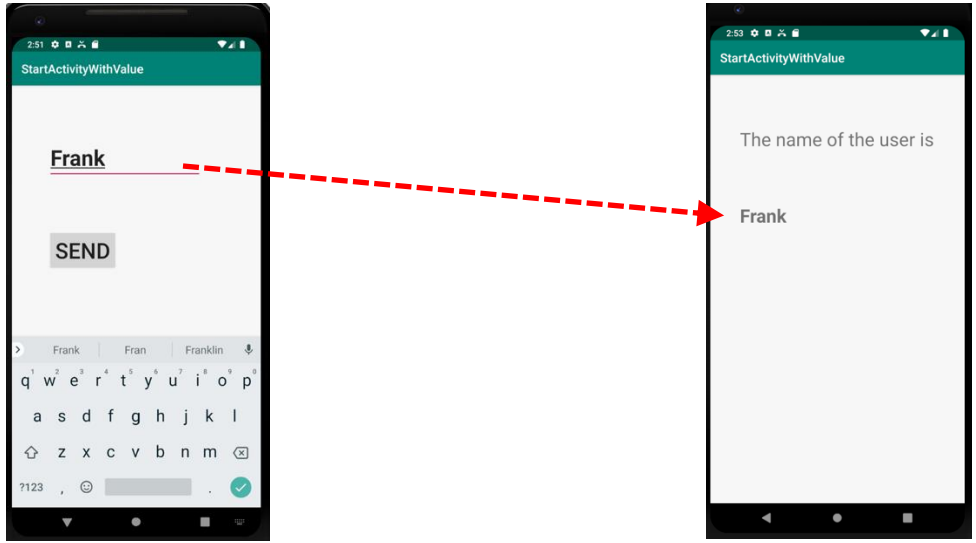
```
fun viewNoteDetail() {
    val intent = Intent(this, NoteDetailActivity::class.java)
    intent.putExtra(NOTE_ID, note.id)
    startActivity(intent)
}
```

Navigate to a specific external app:

```
fun openExternalApp() {
    val intent = Intent("com.example.workapp.FILE_OPEN")
    if (intent.resolveActivity(packageManager) != null) {
        startActivity(intent)
    }
}
```

Providing values to the started activity

- It is possible to provide some values to the activity that is going to be started
- Information is transferred as <*key, value*> pairs attached to intents



Implicit intent

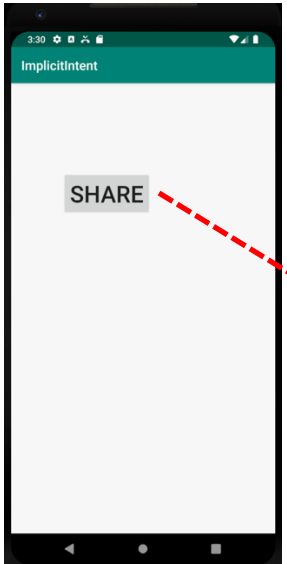
- Provides generic action the app can perform
- Resolved using mapping of the data type and action to known components
- Allows any app that matches the criteria to handle the request

Implicit intent example

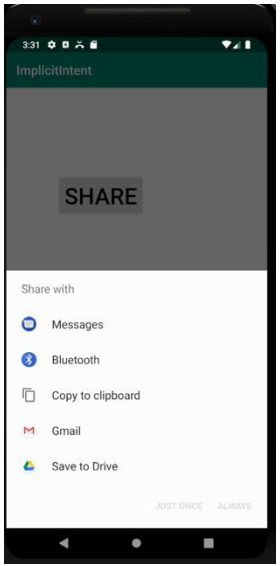
```
fun sendEmail() {
    val intent = Intent(Intent.ACTION_SEND)
    intent.type = "text/plain"
    intent.putExtra(Intent.EXTRA_EMAIL, emailAddresses)
    intent.putExtra(Intent.EXTRA_TEXT, "How are you?")

    if (intent.resolveActivity(packageManager) != null) {
        startActivity(intent)
    }
}
```

Implicit intents

- Typically, many components (apps) can carry out a given action
  - E.g. Many phones have installed multiple apps that can view images
- In this case Android shows a *Chooser*

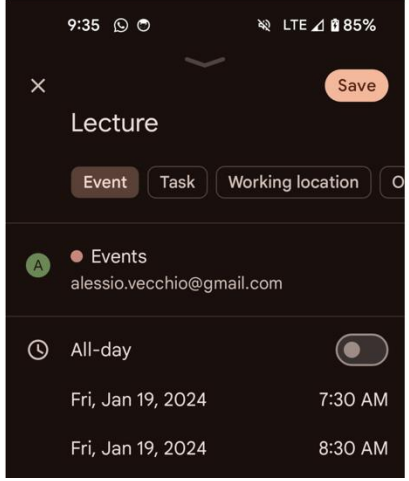



```
fun sendImplicitIntent() {
    val i = Intent(Intent.ACTION_SEND)
    i.setType("text/plain")
    i.putExtra(Intent.EXTRA_TEXT, "Text to be sent")
    startActivity(i)
}
```



Examples of implicit intents

- Setting an alarm
- Add a calendar event
- Take a picture
- View/edit a contact
- Send an email
- Show a location on the map
- Start a phone call
- Open settings




```
override fun onClick(v: View?) {
    val startMillis: Long = Calendar.getInstance().run {
        set(2024, 0, 19, 7, 30)
        timeInMillis
    }
    val endMillis: Long = Calendar.getInstance().run {
        set(2024, 0, 19, 8, 30)
        timeInMillis
    }
    val intent = Intent(Intent.ACTION_INSERT)
        .setData(CalendarContract.Events.CONTENT_URI)
        .putExtra(CalendarContract.EXTRA_EVENT_BEGIN_TIME, startMillis)
        .putExtra(CalendarContract.EXTRA_EVENT_END_TIME, endMillis)
        .putExtra(CalendarContract.Events.TITLE, "Lecture")
        .putExtra(CalendarContract.Events.DESCRIPTION, "MaSSS class")
        .putExtra(CalendarContract.Events.EVENT_LOCATION, "ADI1")
        .putExtra(CalendarContract.Events.AVAILABILITY,
                CalendarContract.Events.AVAILABILITY_BUSY)
    startActivity(intent)
}
```

Broadcast intents

- Some intents are broadcasted by Android to notify apps of system events
- Examples
  - Boot phase completed
  - Wi-Fi state changed
  - Timezone changed
  - Power cord connected/disconnected
  - Power saving mode activated/deactivated
  - …

- For a complete list of system broadcast actions, see the `BROADCAST_ACTIONS.TXT` file in the Android SDK

BroadcastReceiver

- This broadcast receiver is activated when boot is completed:

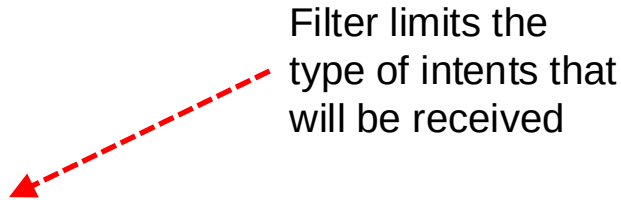
```
class MyReceiver : BroadcastReceiver() {

    override fun onReceive(context: Context, intent: Intent) {
        Log.i("Example", "Boot completed")
    }
}
```

This method is executed by an OS thread: if long running operation you have to start your own threads

BroadcastReceiver

- To receive events, a *BroadcastReceiver* must be registered
- Two options:
  - **Static**: in the manifest file, receives events even if app is not running
  - **Dynamic**: by means of Kotlin code, e.g. in Activities

```
<receiver
    android:name=".MyReceiver"
    android:enabled="true"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action. RECEIVE_BOOT_COMPLETED"/>
    </intent-filter>
</receiver>
```

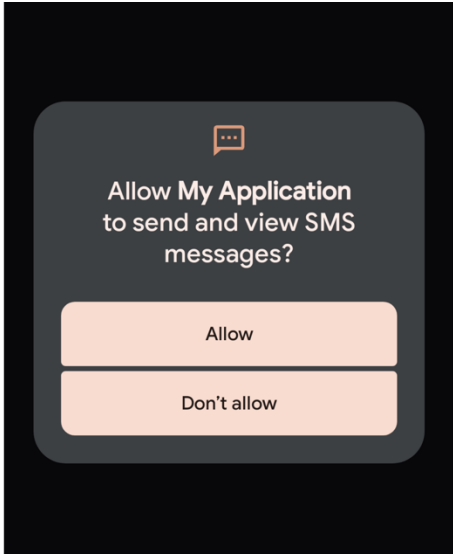


BroadcastReceiver

- Some events requires permissions
- In the manifest:

```
<uses-permission-sdk-23 android:name="android.permission.READ_SMS" />
```

- Some permissions must be requested at **runtime** through Kotlin code
- The number of system broadcasts has been reduced in Android 7+ for improved energy and memory efficiency



Runtime permissions

- Some permissions must be requested at runtime (not installation time)
- Requesting permission needed for the SMS receiver example:

```
private fun requestSmsPermission() {
    when {
        ContextCompat.checkSelfPermission(this, Manifest.permission.RECEIVE_SMS) ==
        PackageManager.PERMISSION_GRANTED -> {
            Log.i(TAG, "permission already granted")
            return
        }
        ActivityCompat.shouldShowRequestPermissionRationale(this, Manifest.permission.RECEIVE_SMS) -> {
            // Explain to the user why your app requires this permission
            Log.i(TAG, "show why")
        }
        else -> {
            // You can directly ask for permission.
            Log.i(TAG, "permission not already granted, I'm asking")
            ActivityCompat.requestPermissions(this, arrayOf(Manifest.permission.RECEIVE_SMS), 1)
        }
    }
}
```

```
override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<String>, grantResults: IntArray) {
    when (requestCode) {
        1 -> {
            // If request is cancelled, the result arrays are empty.
            if ((grantResults.isNotEmpty() && grantResults[0] == PackageManager.PERMISSION_GRANTED)) {
                Log.i(TAG, "permission now granted")
                // Permission is granted.
            } else {
                Log.i(TAG, "permission denied")
            }
            return
        }
    }
    ...
}
```

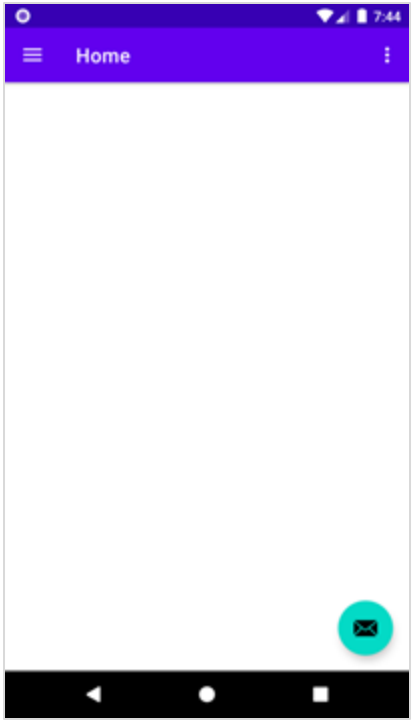
BroadcastReceiver: dynamic registration/unregistration

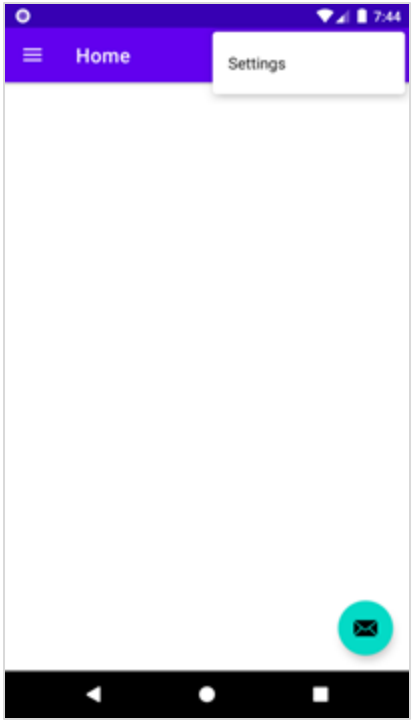
- The same receiver can be registered and unregistered dynamically

```
val TAG = "Example"
val mr = MyReceiver()

 override fun onResume() {
    super.onResume()
    val filter = IntentFilter()
    filter.addAction(Telephony.Sms.Intents.SMS_RECEIVED_ACTION)
    registerReceiver(mr, filter)
    Log.i(TAG, "Registering receiver")
}

override fun onPause() {
    super.onPause();
    unregisterReceiver(mr);
    Log.i(TAG, "Unregistering receiver");
}
```

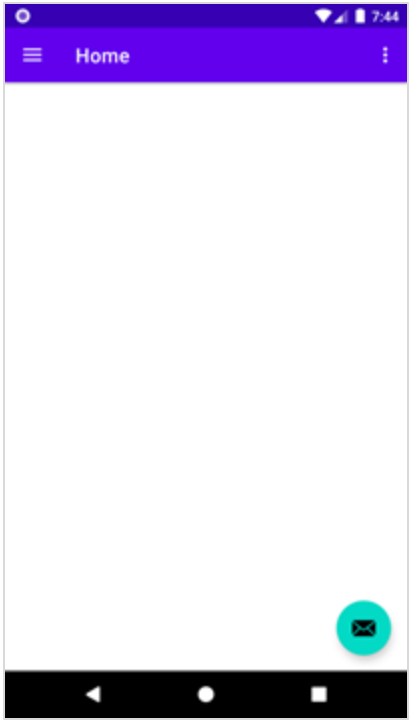
App bar, navigation drawer, and menus

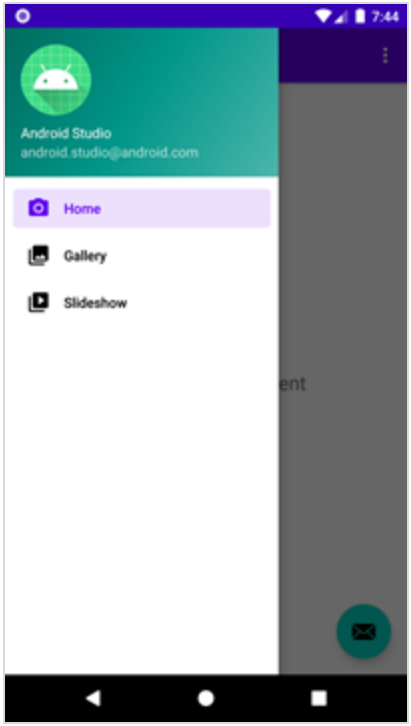
App bar





Navigation drawer





Menu

Define menu items in XML menu resource (located in `res/menu` folder)

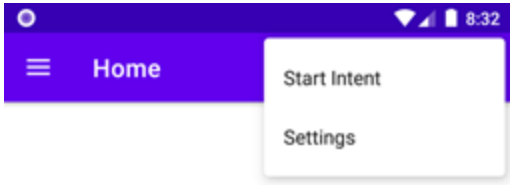
```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/action_settings"
        android:orderInCategory="100"
        android:title="@string/action_settings"
        app:showAsAction="never" />
</menu>
```

More menu options

```
<menu>
    <group android:checkableBehavior="single">
        <item
            android:id="@+id/nav_home"
            android:icon="@drawable/ic_menu_camera"
            android:title="@string/menu_home" />
        <item
            android:id="@+id/nav_gallery"
            android:icon="@drawable/ic_menu_gallery"
            android:title="@string/menu_gallery" />
        <item
            android:id="@+id/nav_slideshow"
            android:icon="@drawable/ic_menu_slideshow"
            android:title="@string/menu_slideshow" />
    </group>

</menu>
```

Options menu example

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <item android:id="@+id/action_intent"
        android:title="@string/action_intent" />

    <item
        android:id="@+id/action_settings"
        android:orderInCategory="100"
        android:title="@string/action_settings"
        app:showAsAction="never" />
</menu>
```



Inflate options menu

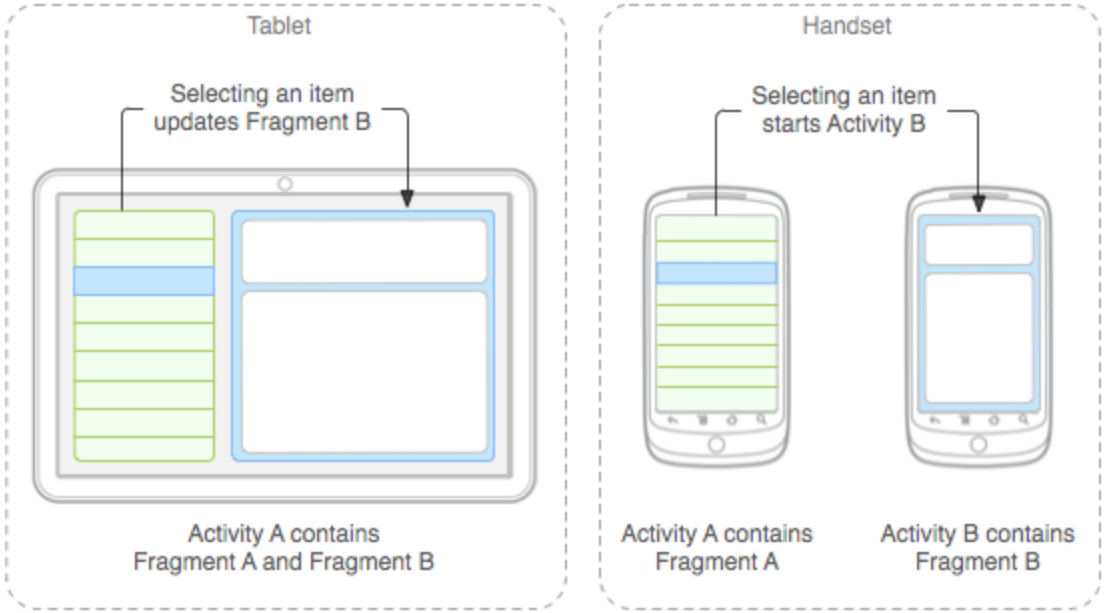
```
override fun onCreateOptionsMenu(menu: Menu): Boolean {
    menuInflater.inflate(R.menu.main, menu)
    return true
}
```

Handle menu options selected

```
override fun onOptionsItemSelected(item: MenuItem): Boolean {

    when (item.itemId) {
        R.id.action_intent -> {
            val intent = Intent(Intent.ACTION_WEB_SEARCH)
            intent.putExtra(SearchManager.QUERY, "pizza")
            if (intent.resolveActivity(packageManager) != null) {
                startActivity(intent)
            }
        }

        else -> Toast.makeText(this, item.title, Toast.LENGTH_LONG).show()
    ...
```

Fragments

Fragments for tablet layouts



Fragment

- Represents a behavior or portion of the UI in an activity ("microactivity")
- Must be hosted in an activity
- Lifecycle tied to host activity's lifecycle
- Can be added or removed at runtime

Note about fragments

Use the AndroidX version of the `Fragment` class. (`androidx.fragment.app.Fragment`).

Don't use the platform version of the `Fragment` class (`android.app.Fragment`), which was deprecated.

Navigation within an app

Navigation component

- Collection of libraries and tooling, including an integrated editor, for creating navigation paths through an app
- Assumes one `Activity` per graph with many `Fragment` destinations
- Consists of three major parts:
  - Navigation graph
  - Navigation Host (`NavHost`)
  - Navigation Controller (`NavController`)

Add dependencies

In `build.gradle`, under `dependencies`:

```
implementation "androidx.navigation:navigation-fragment-ktx:$nav_version"

implementation "androidx.navigation:navigation-ui-ktx:$nav_version"
```

Navigation host (NavHost)

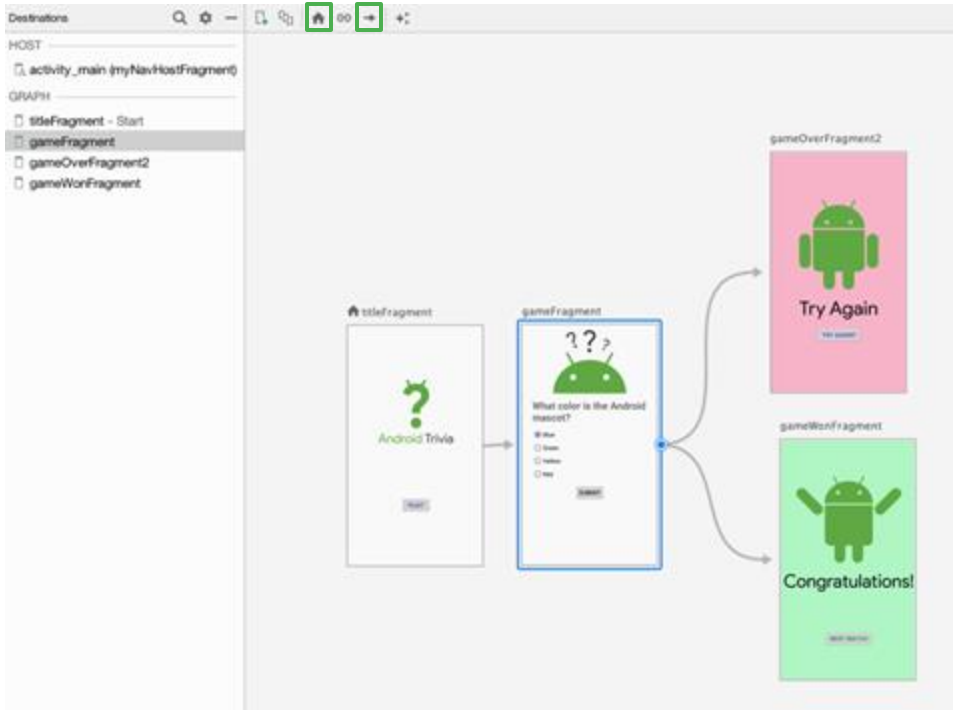
```
<fragment
    android:id="@+id/nav_host"
    android:name="androidx.navigation.fragment.NavHostFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:defaultNavHost="true"
    app:navGraph="@navigation/nav_graph_name"/>
```

Navigation graph

New resource type located in `res/navigation` directory

- XML file containing all of your navigation destinations and actions
- Lists all the (Fragment/Activity) destinations that can be navigated to
- Lists the associated actions to traverse between them
- Optionally lists animations for entering or exiting

Navigation Editor in Android Studio



Creating a Fragment

- Extend `Fragment` class
- Override `onCreateView()`
- Inflate a layout for the Fragment that you have defined in XML

```
class DetailFragment : Fragment() {

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
            savedInstanceState: Bundle?): View? {
        return inflater.inflate(R.layout.detail_fragment, container, false)
    }
}
```

Specifying Fragment destinations

- Fragment destinations are denoted by the `action` tag in the navigation graph.
- Actions can be defined in XML directly or in the Navigation Editor by dragging from source to destination.
- Autogenerated action IDs take the form of `action_<sourceFragment>_to_<destinationFragment>`.

Example fragment destination

```
<fragment
    android:id="@+id/welcomeFragment"
    android:name="com.example.android.navigation.WelcomeFragment"
    android:label="fragment_welcome"
    tools:layout="@layout/fragment_welcome" >

    <action
        android:id="@+id/action_welcomeFragment_to_detailFragment"
        app:destination="@id/detailFragment" />

</fragment>
```

Navigation Controller (NavController)

`NavController` manages UI navigation in a navigation host.

- Specifying a destination path only names the action, but it doesn't execute it.
- To follow a path, use `NavController`.

Example NavController

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        ...
        val navController = findNavController(R.id.myNavHostFragment)
    }

    fun navigateToDetail() {
        navController.navigate(R.id.action_welcomeFragment_to_detailFragment)
    }
}
```

More custom navigation behavior

Passing data between destinations

Using Safe Args:

- Ensures arguments have a valid type
- Lets you provide default values
- Generates a `<SourceDestination>Directions` class with methods for every action in that destination
- Generates a class to set arguments for every named action
- Generates a `<TargetDestination>Args` class providing access to the destination's arguments

Setting up Safe Args

In the project `build.gradle` file:

```
buildscript {
    repositories {
        google()
    }
    dependencies {
        classpath "androidx.navigation:navigation-safe-args-gradle-plugin:$nav_version"
    }
}
```

In the app's or module's `build.gradle` file:

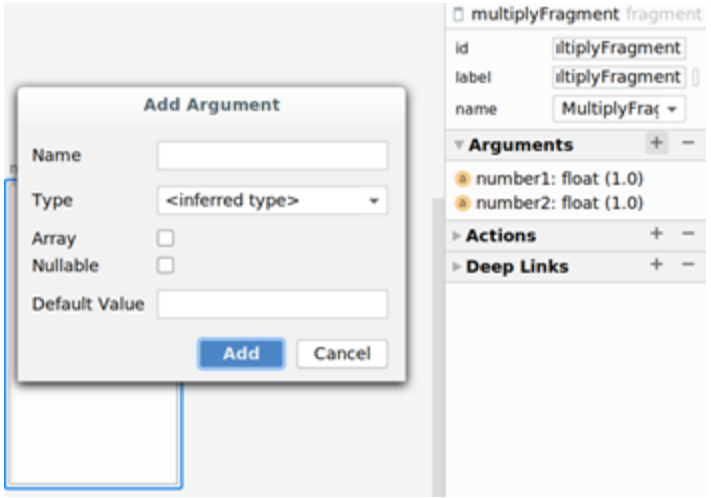
```
apply plugin: "androidx.navigation.safeargs.kotlin"
```

Sending data to a Fragment

1. Create arguments the destination fragment will expect.
2. Create action to link from source to destination.
3. Set the arguments in the action method on `<Source>FragmentDirections`.
4. Navigate according to that action using the Navigation Controller.
5. Retrieve the arguments in the destination fragment.

Destination arguments

```
<fragment
    android:id="@+id/multiplyFragment"
    android:name="com.example.arithmetic.MultiplyFragment"
    android:label="MultiplyFragment" >
    <argument
        android:name="number1"
        app:argType="float"
        android:defaultValue="1.0" />
    <argument
        android:name="number2"
        app:argType="float"
        android:defaultValue="1.0" />
</fragment>
```



Supported argument types

Type	**Type Syntax** `app:argType=<type>`	**Supports Default Values**	**Supports Null Values**
Integer	`"integer"`	Yes	No
Float	`"float"`	Yes	No
Long	`"long"`	Yes	No
Boolean	`"boolean"`	Yes (`"true"` or `"false"`)	No
String	`"string"`	Yes	Yes
Array	above type + `"[]"` (for example, `"string[]"` `"long[]"`)	Yes (only `"@null"`)	Yes
Enum	Fully qualified name of the enum	Yes	No
Resource reference	`"reference"`	Yes	No

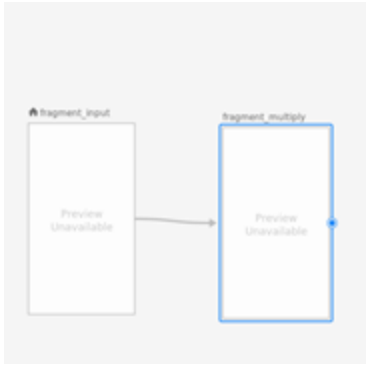
Supported argument types: Custom classes

Type	**Type Syntax** `app:argType=<type>`	**Supports Default Values**	**Supports Null Values**
Serializable	Fully qualified class name	Yes (only `"@null"`)	Yes
Parcelable	Fully qualified class name	Yes (only `"@null"`)	Yes

Create action from source to destination

In `nav_graph.xml`:

```
<fragment
    android:id="@+id/fragment_input"
    android:name="com.example.arithmetic.InputFragment">

    <action
        android:id="@+id/action_to_multiplyFragment"
        app:destination="@id/multiplyFragment" />

</fragment>
```



Navigating with actions

In `InputFragment.kt`:

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    binding.button.setOnClickListener {
        val n1 = binding.number1.text.toString().toFloatOrNull() ?: 0.0
        val n2 = binding.number2.text.toString().toFloatOrNull() ?: 0.0

        val action = InputFragmentDirections.actionToMultiplyFragment(n1, n2)
        view.findNavController().navigate(action)
    }
}
```

Retrieving Fragment arguments

```
class MultiplyFragment : Fragment() {
    val args: MultiplyFragmentArgs by navArgs()
    lateinit var binding: FragmentMultiplyBinding
    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        val number1 = args.number1
        val number2 = args.number2
        val result = number1 * number2
        binding.output.text = "${number1} * ${number2} = ${result}"
    }
}
```

Navigation UI

Menus revisited

```
override fun onOptionsItemSelected(item: MenuItem): Boolean {
    val navController = findNavController(R.id.nav_host_fragment)
    return item.onNavDestinationSelected(navController) ||
            super.onOptionsItemSelected(item)
}
```

DrawerLayout for navigation drawer

```
<androidx.drawerlayout.widget.DrawerLayout
    android:id="@+id/drawer_layout" ...>

    <fragment
        android:name="androidx.navigation.fragment.NavHostFragment"
        android:id="@+id/nav_host_fragment" ... />

    <com.google.android.material.navigation.NavigationView
        android:id="@+id/nav_view"
        app:menu="@menu/activity_main_drawer" ... />


</androidx.drawerlayout.widget.DrawerLayout>
```

Finish setting up navigation drawer

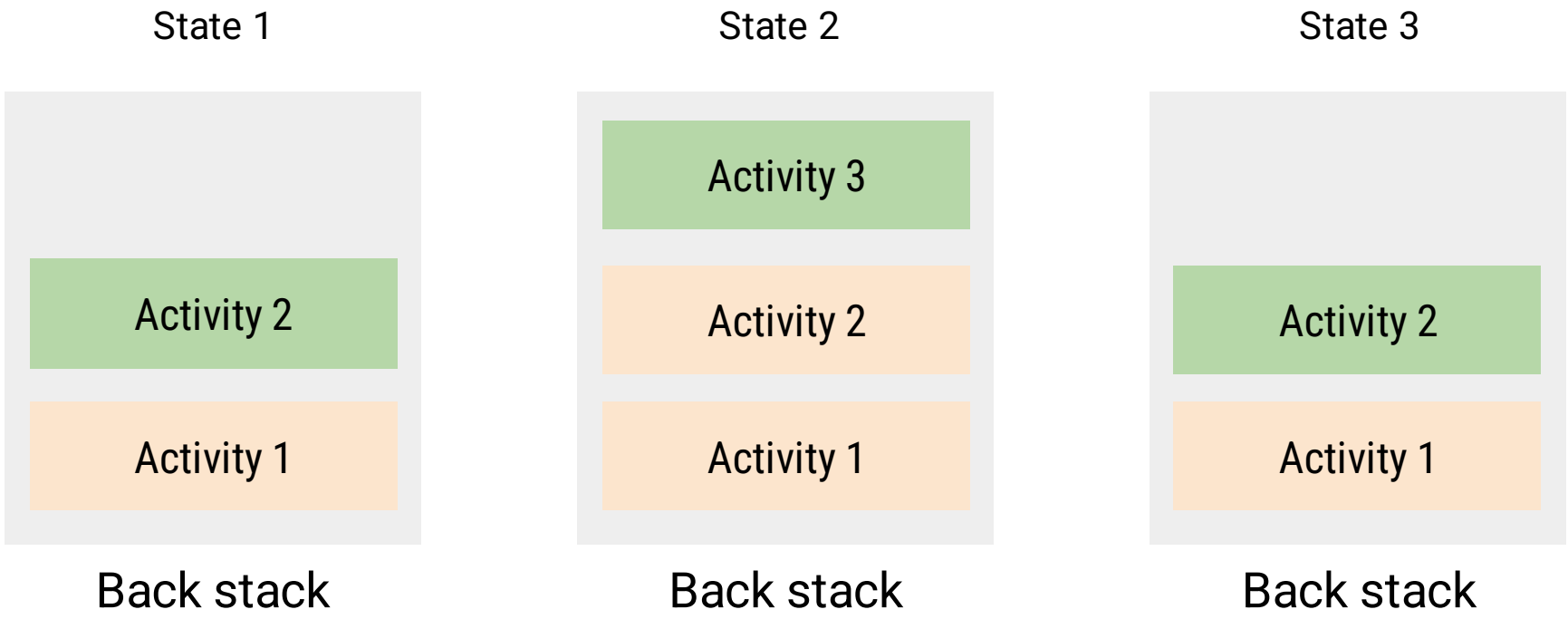
Connect `DrawerLayout` to navigation graph:

```
val appBarConfiguration = AppBarConfig(navController.graph, drawer)
```

Set up `NavigationView` for use with a `NavController`:

```
val navView = findViewById<NavigationView>(R.id.nav_view)
navView.setupWithNavController(navController)
```

Understanding the back stack



Fragments and the back stack