

AD340 Mobile Application Development

...

Week 04

Outline

Data Binding

RecyclerView

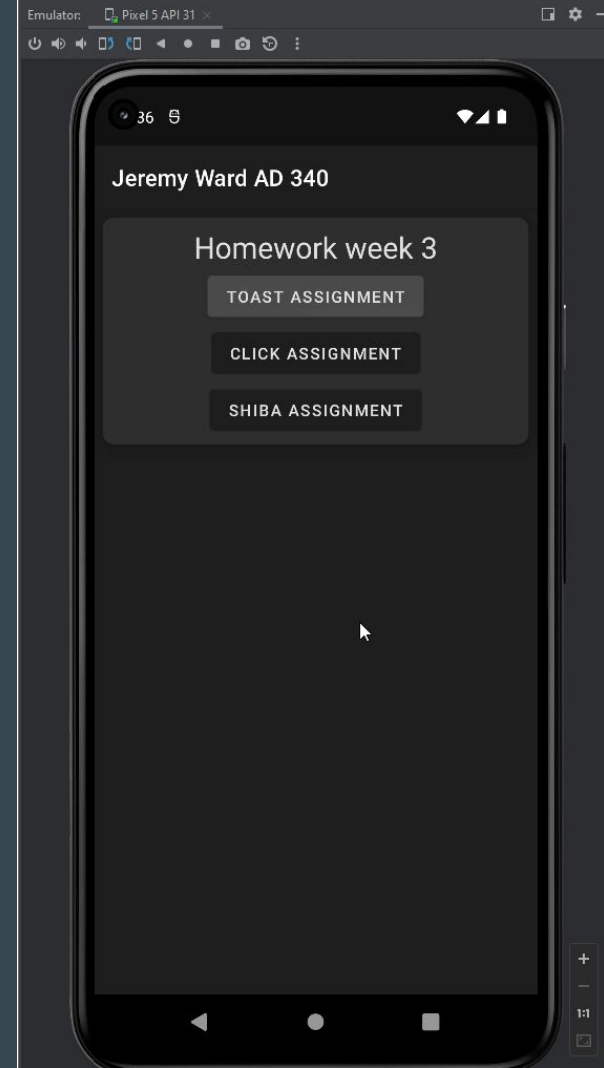
Multiple activities and intents

App Bar, Navigation drawer, and Menus

Fragments

Assignment example

Feel free to submit your app in the Q&A section



Last Week

Make an app interactive

Building an Android App

Accessibility

Layouts in general

ConstraintLayout

Questions?

Data Binding

Current approach: findViewById()

Within your Activity code, you could use `findViewById()` to find View instances

Issues

- IDs are global to all layouts
- runtime crashes if you reference an ID that isn't in the current layout

MainActivity.kt

```
val name = findViewById(...)
val age = findViewById(...)
val loc = findViewById(...)

name.text = ...
age.text = ...
loc.text = ...
```

activity_main.xml

```
<ConstraintLayout ... >
  <TextView
    android:id="@+id/name"/>
  <TextView
    android:id="@+id/age"/>
  <TextView
    android:id="@+id/loc"/>
</ConstraintLayout>
```

findViewById

findViewById

findViewById

Use data binding instead

Why

- simplify your code
- catch bugs at compile time by having type safety with the views you're accessing

MainActivity.kt

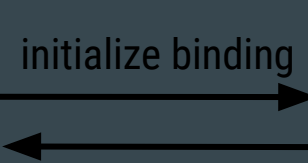
```
Val binding:ActivityMainBinding
```

```
binding.name.text = ...
```

```
binding.age.text = ...
```

```
binding.loc.text = ...
```

initialize binding



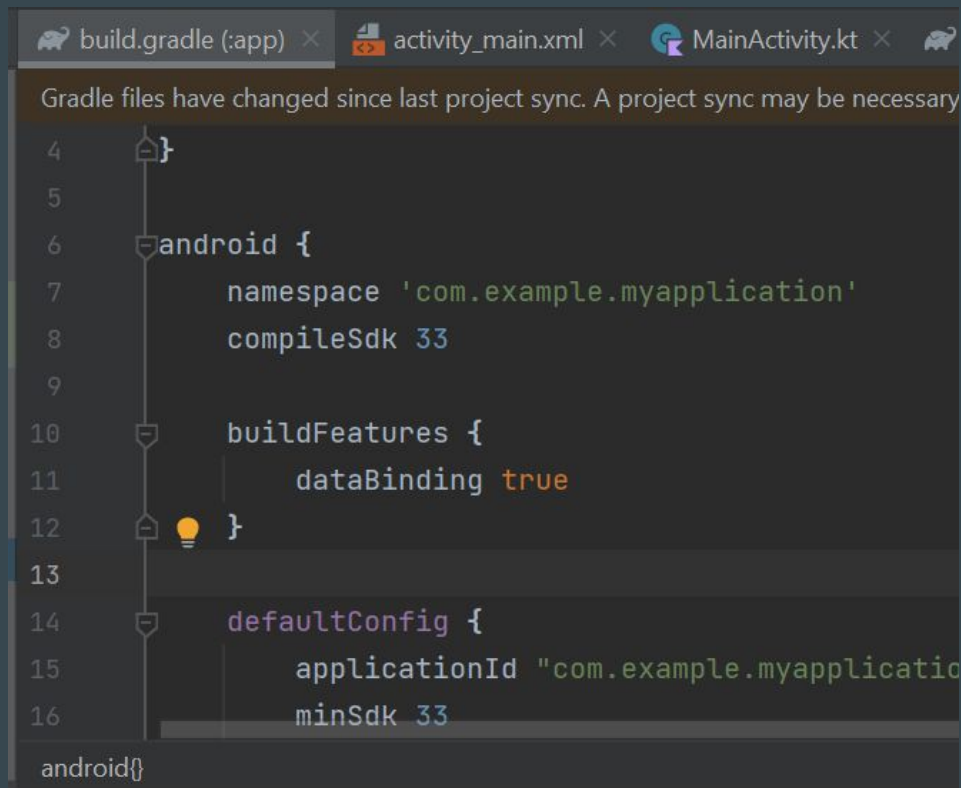
activity_main.xml

```
<layout>  
  <ConstraintLayout ... >  
    <TextView  
      android:id="@+id/name"/>  
    <TextView  
      android:id="@+id/age"/>  
    <TextView  
      android:id="@+id/loc"/>  
  </ConstraintLayout>  
</layout>
```


Modify build.gradle file

enable the dataBinding build option in your build.gradle file in the app module

it should generate some classes for us when we mark a layout in XML



The screenshot shows an IDE window with three tabs: 'build.gradle (:app)', 'activity_main.xml', and 'MainActivity.kt'. The 'build.gradle (:app)' tab is active, displaying the following Kotlin code:

```
4  }
5
6  android {
7      namespace 'com.example.myapplication'
8      compileSdk 33
9
10     buildFeatures {
11         dataBinding true
12     }
13
14     defaultConfig {
15         applicationId "com.example.myapplication"
16         minSdk 33
17     }
18 }
```

A notification bar at the top of the editor states: "Gradle files have changed since last project sync. A project sync may be necessary". A yellow lightbulb icon is visible next to the closing brace of the 'buildFeatures' block on line 12.

Add layout tag

indicating that we want a binding class created for us

The binding class will contain references to any views that have a resource ID

```
<layout>
    <androidx.constraintlayout.widget.ConstraintLayout
        xmlns:app="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".MainActivity">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello World!"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent"

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

Layout inflation with data binding

instead of calling `setContentView()` from the Activity

call `DataBindingUtil.setContentView()`

- pass the Activity(this)
- resource ID of the desired layout

We didn't have to call `findViewById!`

```
import androidx.databinding.DataBindingUtil
import com.example.data_binding_example_01.databinding.ActivityMainBinding

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        val binding: ActivityMainBinding = DataBindingUtil.setContentView(
            activity: this, R.layout.activity_main)

        binding.textview01.setOnClickListener { it: View!
            binding.textview01.text = "goodbye"
        }
    }
}
```

Data binding layout variables

Instead of modifying the binding values in code, we can set them in the layout

insert the data tag to declare that the layout will have access to variables

```
class MainActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        val binding: ActivityMainBinding = DataBindingUtil.  
            activity: this, R.layout.activity_main  
        binding.name = "Hello World!"  
        binding.textview01.setOnClickListener {  
            binding.name = "goodbye"  
        }  
    }  
}
```

```
<data>  
    <variable name="name" type="String"/>  
</data>  
<androidx.constraintlayout.widget.ConstraintLayout  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    tools:context=".MainActivity">  
  
    <TextView  
        android:id="@+id/textview01"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="@{name}"  
        app:layout_constraintBottom_toBottomOf="parent"  
        app:layout_constraintEnd_toEndOf="parent"  
        app:layout_constraintStart_toStartOf="parent"  
        app:layout_constraintTop_toTopOf="parent" />
```

Data binding layout expressions

Expression language lets you do many things in a single line of code

- simple transformations
- displaying content based on a comparison
- accessing the content of other views

Example

- Call toUpperCase() on the name variable String

```
<data>
    <variable name="name" type="String"/>
</data>

<androidx.constraintlayout.widget.ConstraintLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textview01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@{name.toUpperCase()}"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

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

Displaying lists with RecyclerView

What is RecyclerView?

Widget for displaying lists of data

Recycles (reuses) item views to make scrolling more performant

Can specify a list item layout for each item in the dataset

Supports animations and transitions

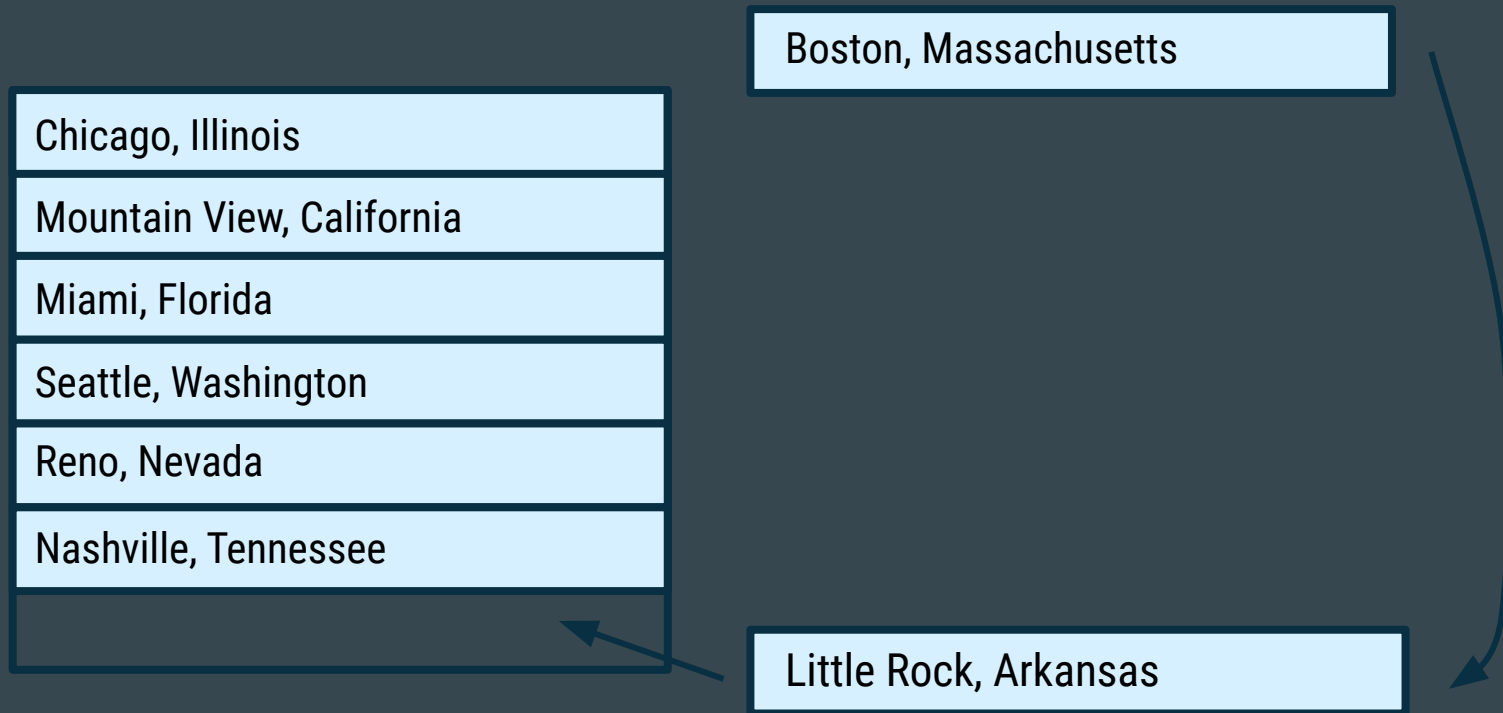
RecyclerView.Adapter

A basic RecyclerView adapter will need to override the following three functions:

- **getItemCount** returns the total number of items available in your list of data
- **onCreateViewHolder** is called to create a new list item layout
- **onBindViewHolder** is called when reusing a list item layout by updating the data that's displayed within it

A ViewHolder represents a list item layout and has references to all the views within the list item layout.

View recycling in RecyclerView

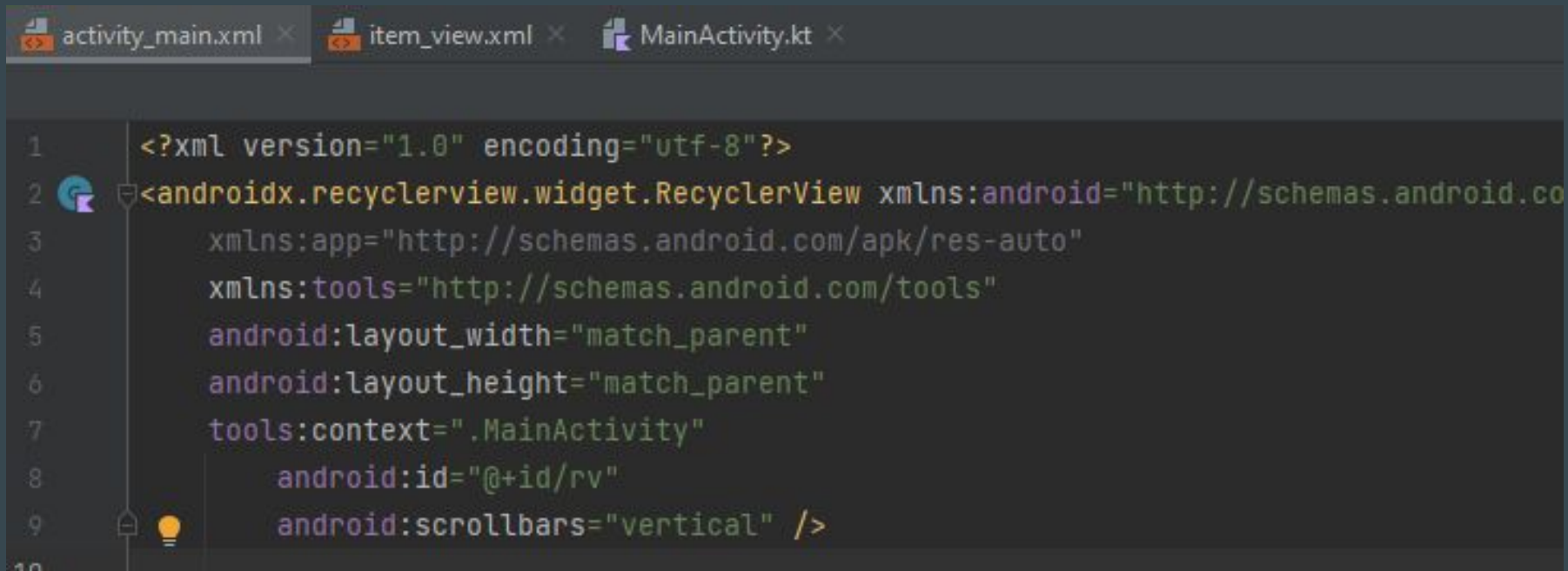


View recycling in RecyclerView

When you have a large number of items in your list, don't create a View for items that have not been scrolled to yet.

- `onCreateViewHolder()` called → view holders for the number of items that can be displayed
- when you scroll, the system removes offscreen list item views from the hierarchy, and calls `onBindViewHolder()`
- values within the list item view are updated to reflect the data in the new list item

Add RecyclerView to your layout



The screenshot shows an IDE with three tabs: `activity_main.xml`, `item_view.xml`, and `MainActivity.kt`. The `activity_main.xml` tab is active, displaying the following XML code:

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.recyclerview.widget.RecyclerView xmlns:android="http://schemas.android.co
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      tools:context=".MainActivity"
8      android:id="@+id/rv"
9      android:scrollbars="vertical" />
```

Line 2 features a blue circular icon with a white 'G' and a purple square icon. Line 9 has a yellow lightbulb icon. The code is color-coded: XML tags are green, attributes are purple, and values are grey.

Create a list item layout

Create a new layout file that represents a list item layout

This list item layout is used for each entry in the list.

```
activity_main.xml × item_view.xml × MainActivity.kt ×
1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res-auto"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      android:layout_width="match_parent"
5      android:layout_height="match_parent">
6      <FrameLayout
7          android:layout_width="wrap_content"
8          android:layout_height="wrap_content"
9          app:layout_constraintBottom_toBottomOf="parent"
10         app:layout_constraintEnd_toEndOf="parent"
11         app:layout_constraintStart_toStartOf="parent"
12         app:layout_constraintTop_toTopOf="parent">
13
14         <TextView
15             android:id="@+id/number"
16             android:layout_width="match_parent"
17             android:layout_height="wrap_content"
18             android:textSize="34sp" />
19     </FrameLayout>
20 </androidx.constraintlayout.widget.ConstraintLayout>
```

Create a list adapter

```
class MyAdapter(val data: List<Int>) : RecyclerView.Adapter<MyAdapter.MyViewHolder>() {  
    class MyViewHolder(val row: View) : RecyclerView.ViewHolder(row) {  
        val textView = row.findViewById<TextView>(R.id.number)  
    }  
  
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MyViewHolder {  
        val layout = LayoutInflater.from(parent.context).inflate(R.layout.item_view,  
            parent, attachToRoot: false)  
        return MyViewHolder(layout)  
    }  
  
    override fun onBindViewHolder(holder: MyViewHolder, position: Int) {  
        holder.textView.text = data.get(position).toString()  
    }  
  
    override fun getItemCount(): Int = data.size  
}
```

Set the adapter on the RecyclerView

```
class MainActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
  
        val rv: RecyclerView = findViewById(R.id.rv)  
        rv.layoutManager = LinearLayoutManager(context, this)  
  
        rv.adapter = MyAdapter(IntRange(0, 100).toList())  
    }  
}
```

Multiple activities and intents

Multiple screens in an app

We've looked at Android apps that have a single screen that is implemented as a single Activity

More functionality → separate the features into different screens

Implement as individual Activities with a specific purpose.

Why?

- Code maintenance
- Code reuse

Multiple screens in an app: 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)

Intent

Android uses a construct called an Intent to request an action from another component

Example

- an Intent can specify a request to transition to another Activity
- An Intent could contain data for the destination Activity to use
 - Details about an item to be displayed
- Data can also be passed back to the source activity
- For now, we will focus on navigating between Activities.

Types of Intents

Two primary types of intents

- Explicit Intent
- Implicit Intent

Explicit Intent

Strict type: indicates a specific component that should handle the request

Commonly used when navigating between components within your app

- Can navigate to a known third-party app.
- Only if you know the type of intent they can handle

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)  
}
```

a second intent that navigates to an external app

first check that we can resolve the Intent;

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

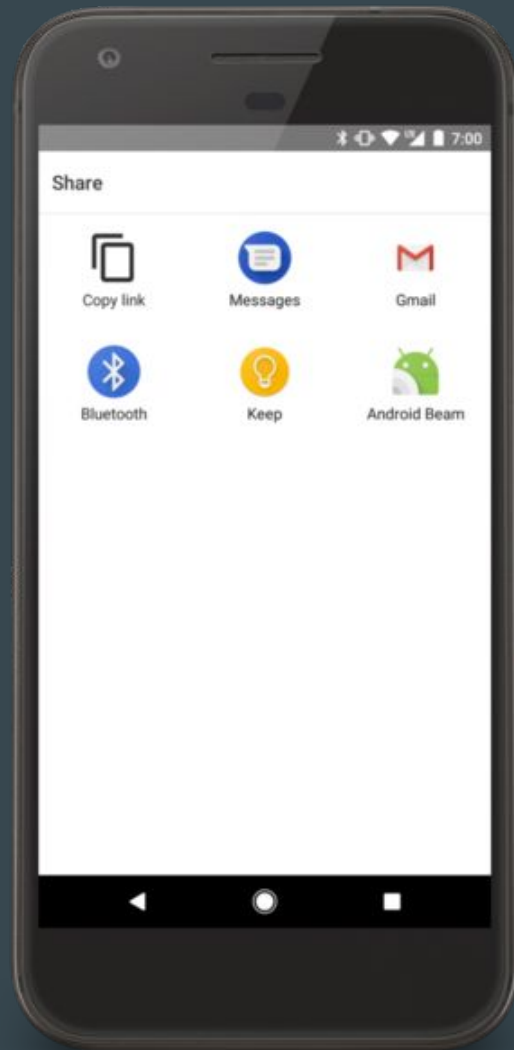
Implicit intent

Implicit intents on the other hand don't specify an intended target

Provide just enough information for the system to resolve

Implicit intent is recommended for components that your app doesn't own.

If multiple apps can handle the intent, the system lets the user select one



Implicit intent example

creating an implicit intent to send an email

don't mind which email app handles the request

```
fun sendEmail() {  
    val intent = Intent(Intent.ACTION_SEND)  
    intent.type = "text/plain"  
    intent.putExtra(Intent.EXTRA_EMAIL, emailAddresses)  
    intent.putExtra(Intent.EXTRA_TEXT, value: "How are you?"  
  
    if (intent.resolveActivity(packageManager) != null) {  
        startActivity(intent)  
    }  
}
```

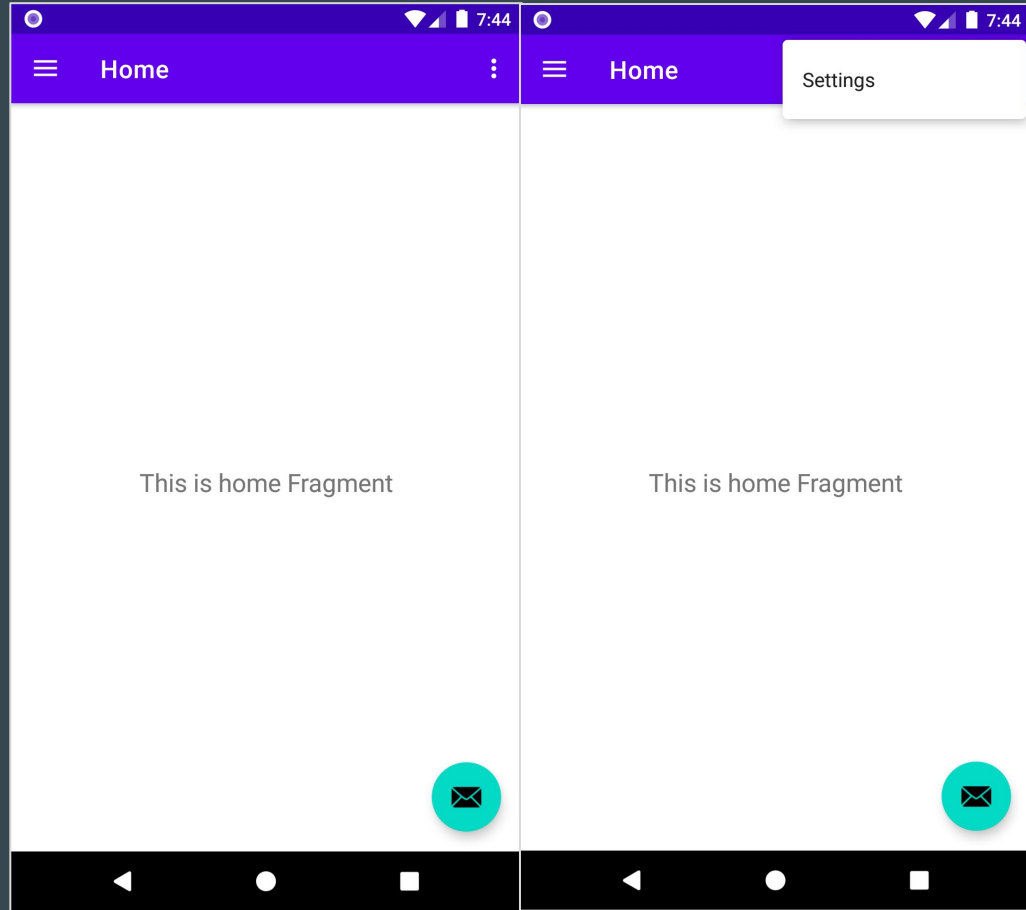
App Bar, Navigation drawer, and Menus

App bar

displays the name of your Activity or app

provides the user with access to important actions

- overflow menu (3 dots)
- support for navigation (hamburger menu)
- view switching (with tabs or drop-down lists)

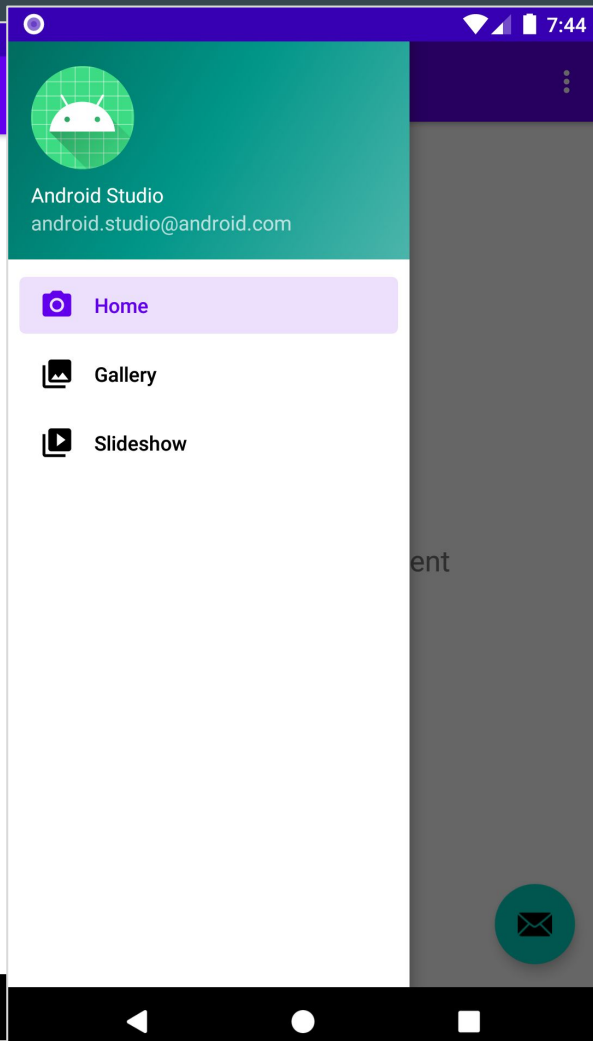
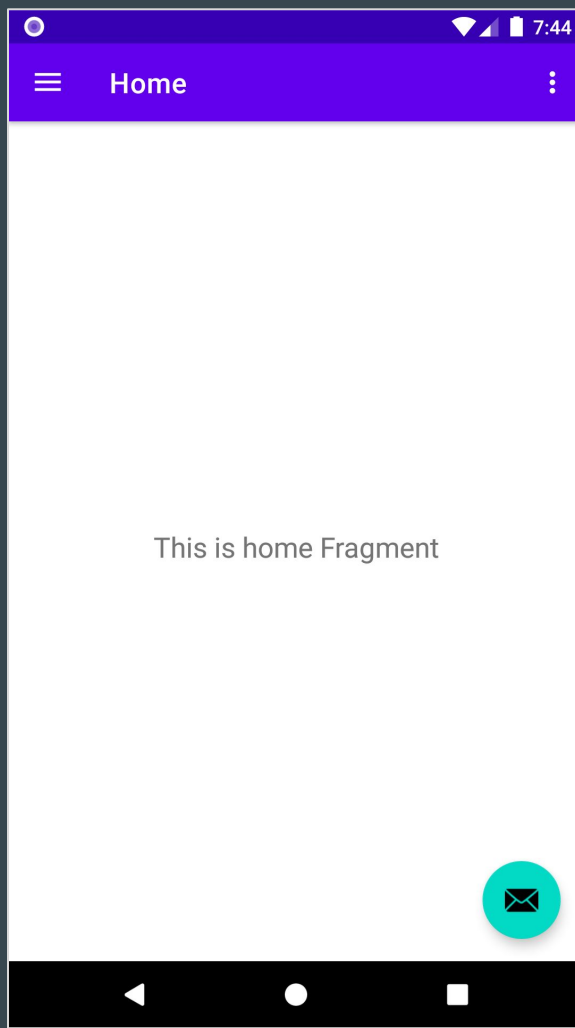


Navigation Drawer

an element that's often seen in apps

- Open the drawer by tapping the hamburger icon
- swipe in gesture from the left side of the screen

lets you quickly navigate to locations in your app



Menus

Different types of menus

- options menus
- context menus
- popup menus

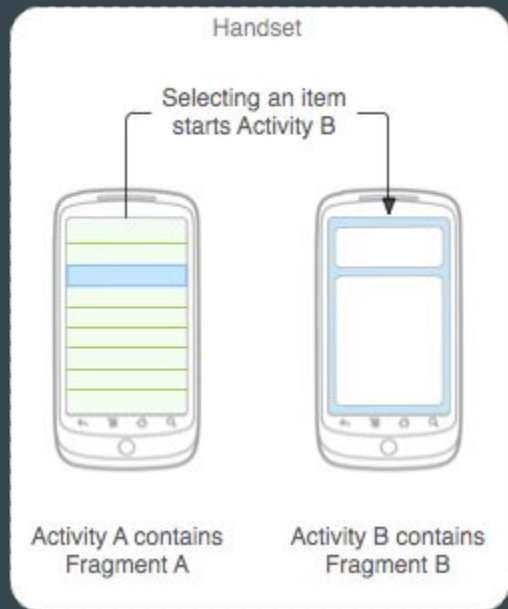
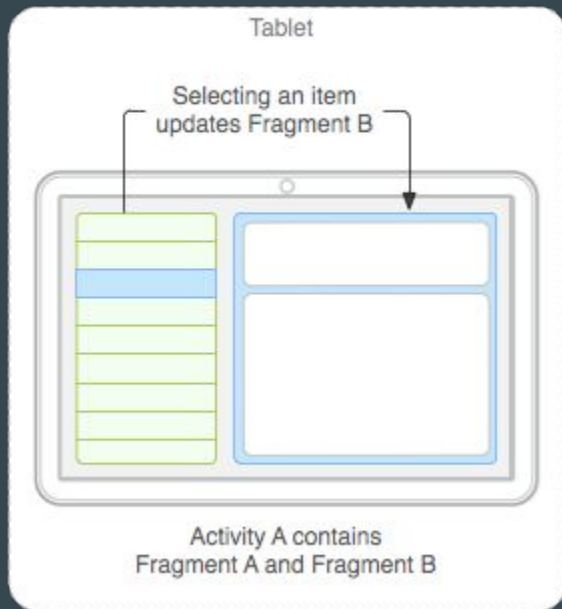
Fragments

Fragments for layouts

When we navigate to different screens in our app, we can use different activities. But we can also use fragments.

First introduced when support for tablets was added.

breaking up the structure of your UI into fragments is useful in other contexts besides building for tablets



Fragment

represents a behavior or a portion of the UI in an Activity

think of it as a "micro activity"

- Can display multiple fragments in a single activity (tablet case)
- Reuse a fragment in multiple activities
- Has its own lifecycle
- Receives its own input events
- Can add or remove while the activity is running

Two Fragment classes

Platform version: `android.app.Fragment`

AndroidX version: `androidx.fragment.app.Fragment`

- Be sure to use the AndroidX version.
- Platform version is deprecated in API level 28