Lecture 3: Multi-view UI & Android Application Components
So far

- Activity lifecycle
- Activity design
  - Layouts, Widgets (Viewgroups & Views)
- Activity interactivity
UI components – pt. 3
Support & Compatibility libraries

• We’ve seen how the framework has evolved over the years
  • New features, deprecated features
    • e.g. RelativeLayout → ConstraintLayout

• Look & Feel of Android in general has also changed
  • [Material design](https://material.io) introduced in API 21

• Potentially, this could lead to disparities in UX across versions

https://developer.android.com/topic/libraries/support-library/
Notifications evolution example
Support & Compatibility libraries

• Some API features don’t exist on older versions
  • E.g. ConstraintLayout supported for API v9+ (2010),
    although ConstraintLayout introduced in 2016

• Some modern UI features are packaged as a Support library
  • E.g. Floating action button
    • android.support.design.widget.FloatingActionButton

• Convenience and Helper classes
  • E.g. Recycler VIA – manage large list views

https://developer.android.com/topic/libraries/support-library/
Using support libraries

• Default project configurations already include them!

• AndroidX
  • Contains the support libraries
  • A good starting overview of contained features can be found here:
    • https://developer.android.com/topic/libraries/support-library/features

https://developer.android.com/jetpack/androidx
Toolbar (App Bar / Action Bar)

- A dedicated space for familiar user interaction
- Access to important actions in a quick, predictable way
- Common uses:
  - Show current location/state within the App
  - Search
  - Additional settings
  - Navigation & view switching (with tabs or drop-down lists)

https://developer.android.com/training/appbar/
Setting up a Toolbar

*By default, projects set up an ActionBar*
- Has different native behaviour depending on versions.
- We will override Actionbar with Toolbar
  - Create Activity as an extension of AppCompatActivity
- Set the app theme to one of appcompat’s “NoActionBar” themes in Manifest
  - E.g. `@style/Theme.AppCompat.Light.NoActionBar`
- Set the toolbar to act as actionbar in Activity onCreate()
  - `setSupportActionBar(findViewById(R.id.my_toolbar))`
Adding Action Buttons

• Define items in a XML resource in res/menu (Action Views)

```xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@id/new_game"
       android:icon="@drawable/ic_new_game"
       android:title="@string/new_game"
       android:showAsAction="ifRoom"/>
  <item android:id="@id/help"
       android:icon="@drawable/ic_help"
       android:title="@string/help" />
</menu>
```

• Override onCreateOptionsMenuMenu

```kotlin
override fun onCreateOptionsMenuMenu(menu: Menu?): Boolean {
    menuInflater.inflate(R.menu.bar_buttons, menu)
    return true
}
```

https://developer.android.com/training/appbar/actions
Defining items

• Basic items behave as buttons, with icons/text
  • app:actionLayout – specify special layout for item (e.g. switch)
  • app:actionViewClass – specify special behavior for item e.g. Searchview

• android:showAsAction – whether it appears as a separate button
  • always, never, ifRoom, withText
  • ifRoom|withText
  • ifRoom|collapseActionView
  • collapseActionView – for dynamic actionviews
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://..." xmlns:app="http://...">
    <item
        android:id="@+id/bar_new"
        android:icon="@android:drawable/ic_menu_add"
        android:title="@string/new_string"
        app:showAsAction="ifRoom|withText"/>
    <item
        android:id="@+id/app_bar_search"
        android:icon="@drawable/ic_search_black_24dp"
        android:title="@string/search"
        app:showAsAction="ifRoom|collapseActionView"
        app:actionViewClass="android.widget.SearchView" />
    <item
        android:id="@+id/bar_help"
        android:title="@string/help_string"
        app:showAsAction="ifRoom"/>
    <item
        android:id="@+id/settings"
        android:title="@string/settings"
        app:showAsAction="never"/>
</menu>
Handling toolbar interaction

• Override the behaviour in Activity:

```kotlin
override fun onOptionsItemSelected(item: MenuItem): Boolean {
    when (item.itemId) {
        R.id.bar_newgame -> {
            // Start a new game
            return true
        }
        R.id.bar_help -> {
            // Show help activity
            return true
        }
        else -> { return super.onOptionsItemSelected(item) }
    }
}
```
Other uses of Menu resources

Navigation Drawer

Bottom Navigation

Pop-up messages
Toasts

- short-length and short-lived text messages
- Use for feedback – e.g. “Sending e-mail...”

```kotlin
val text = "This is a toast message!"
val duration = Toast.LENGTH_SHORT

val toast = Toast.makeText(this, text, duration)
toast.show()
```

- Can also specify position with `toast.setGravity()`

https://developer.android.com/guide/topics/ui/notifiers/toasts
https://developer.android.com/training/snackbar
Snackbar pop-up message

- Snackbar has been introduced as a successor to Toasts
  - More customization – e.g. buttons and actions
- Recommended to use with a CoordinatorLayout
  1) Wrap your existing UI inside a CoordinatorLayout
  2) Refer to the CoordinatorLayout in code
  3) Use setAction(...) to add actions

```java
Snackbar.make(findViewById(R.id.coordinator_layout),
"This is a snackbar message",
Snackbar.LENGTH_LONG)
 .show()
```

https://developer.android.com/training/snackbar/showing.html
Fragments
Fragments

- Fragment – a portion of the UI
- Dynamic, multi-pane user interfaces:
  - With larger screens, there’s more room to combine, interchange, re-use UI elements
- Fragments manage this for you
- Fragments are hosted within activities, thus they are also affected by the host activity's lifecycle!

https://developer.android.com/guide/components/fragments
Creating Fragments

• Create a Fragment subclass, override at least `onCreateView`
  • Also `onCreate()`, `onPause()`

```kotlin
class MyFragment : Fragment() {

  override fun onCreateView(
    inflater: LayoutInflater,

    // Inflate the layout for this fragment
    return inflater.inflate(R.layout.my_fragment, container, false)
  }
}
```

• Include the fragment in an Activity’s Layout
  • E.g. landscape mode or tablet mode

```xml
<LinearLayout>
  <TextView="Hello World" />

  <fragment
    android:id="@+id/fragment_view"
    android:name="example.com.app.MyFragment"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
</LinearLayout>
```

https://developer.android.com/training/basics/fragments/creating
Programmatic fragment transactions

- FragmentManager
  - Fetching existing fragments
    - `findFragmentById()`, `findFragmentByTag()`
  - `add()`, `remove()`, `replace()`
  - `add()`:  
    - 1st argument – ViewGroup into which to place the Fragment  
    - 2nd argument- instance of the fragment

```kotlin
val fragmentManager = supportFragmentManager
val transaction = fragmentManager.beginTransaction()
transaction.add(R.id.container_layout, MyFragment())
transaction.commit()
```

https://developer.android.com/guide/components/fragments#Transactions
Activity-Fragment communication

From Fragment:
- `getActivity()` retrieves the parent Activity instance

From Activity:
- `supportFragmentManager.findFragmentById(..)`

```java
val fragment = supportFragmentManager.findFragmentById(R.id.example_fragment)
```
Fragment Navigation

• A common approach is Single-Activity Architecture
• One Activity hosts all views as Fragments
• To support this, Navigation Component is

Application Components
Basic Application Components

- Essential building blocks of an Android application
- Entry points for system or user to enter your app

• **Activities** – interacting with user

• **Services**
  • Functionality that runs in the background
  • May be called by other components such as activities or services

• **Broadcast Receivers**
  • Enable delivering events from outside regular user flow
  • E.g. system-wide broadcast announcements
  • Avoids keeping an app running to wait for an event (e.g. battery low, screen off)
  • Apps can also initiate Broadcasts

• **ContentProviders**
  • Enable applications to share data, incl. reading, writing, modifying
  • E.g. Phone Contacts

https://developer.android.com/guide/components/fundamentals#Components
Starting Components - Intents

• Recall how applications run in their sandboxes
  • Isolated from each other

• In Android, any app can start another app’s component.
  • E.g. Camera
  • My App -> Camera App -> My App

• **Intents** are used to activate 3 of the main component types:
  • Activities, Services and Broadcast receivers.

• Asynchronous

Working with Intents

• Declare the details of which component you wish to invoke by creating an **Intent** object

• Send out the Intent message, with:
  • startActivity(intent);
  • startService(intent);
  • sendBroadcast(intent);
  • ... *and others*

• System matches Intent with components that match the Intent description (e.g. *intent to capture a photo*)

• Activities and BroadcastReceivers describe what Intents they can service in their **IntentFilters** (via AndroidManifest.xml)
Example: Start an Activity

• Create an Intent object
  1) Provide the current Context (activity, this)
  2) Specify the activity class you want to launch
• Pass Intent to startActivity()

```java
val intent = Intent(this, OtherActivity::class.java)
startActivity(intent)
```

• How do we know whether the activity was actually successfully started or finished?
• What if we want to receive some results?
• How do we pass data?
 Extras

• You can add your own data to the Intent using Extras
  • Intent.putExtra(key, value)

Extra - a Bundle (Key-value pairs) of any additional information

• Can be used to provide extended information to the component
  • Example: if we have an action to send an e-mail message, we could also include extra pieces of data here to supply a subject, body, etc.
Reading value from Bundle

• If Activity A starts Activity B using an Intent
• Activity B can read the Extras put into the Intent:

```
// Activity A:
val intent = Intent(this, OtherActivity::class.java)
intent.putExtra("username", "Jakob")
startActivity(intent)

// Activity B:
val userName = intent.getStringExtra("username")
```
Activity back stack - LIFO

- When an activity is launched, it is put into a back stack
- Hitting the back button pops the activity at the top of the stack
- The set of activities of an app in a stack are referred to as a task
- Hitting Home button doesn’t remove anything from the stack, instead the task stack is sent to background
  - If you then switch to another app (task), that task’s stack is brought to foreground

Retrieving result from Activity

• Activity A starts activity B, user produces a result in data B, then returns to A

• How to get the resulting data produced in B?
  1) In A: `startActivityForResult(..)` instead of `startActivity()`,
     - providing Intent and a `request code as args`
  2) In B:
      ```java
      setResult(resultCode, data)
      finish()
      ```
  3) In A, override:
      ```java
      onActivityResult (requestCode: Int, resultCode: Int, data: Intent)
      ```

https://developer.android.com/training/basics/intents/result
Implicit & Explicit Intents

- **Explicit**
  - Targeted at a specific application by specifying the package name or class name
  - Use it to start components of your own app

- **Implicit**
  - Do not specify the component exactly, but rather a general action

https://developer.android.com/guide/components/intents-filters#Types
<table>
<thead>
<tr>
<th>Component Name</th>
<th>Action</th>
<th>Data</th>
<th>Category</th>
<th>Extras</th>
<th>Flags</th>
</tr>
</thead>
</table>

**Intent**
**Intent**

<table>
<thead>
<tr>
<th>Component Name</th>
<th>Action</th>
<th>Data</th>
<th>Category</th>
<th>Extras</th>
<th>Flags</th>
</tr>
</thead>
</table>

- **Component name**
  - name of the component to start - specify using a fully qualified class name
  - Optional (determines whether implicit or explicit Intent)
- **Action** - The general action to be performed
  - `ACTION_SEND` - “Share”
  - `ACTION_CALL` - Initiate a phone call
  - `ACTION_WEB_SEARCH` - Perform a web search
  - `ACTION_VIEW` - show something to the user (e.g. contact)

• **Data**
  
  • The URI of the data to be acted on

```kotlin
val uri: Uri = Uri.parse("content://contact/people")
```

```kotlin
val uri: Uri = Uri.fromFile(File("/sdcard/cats.jpg"))
intent.setData(uri)
intent.setType("image/jpg")
```

• E.g. if action is ACTION_EDIT, data should contain URI of the document to edit

• In addition, should specify the MIME type of data, not just URI
Examples of action/data pairs

• **ACTION_VIEW** `content://contacts/people/1`
  - Display information about the person with identifier "1"

• **ACTION_DIAL** `content://contacts/people/1`
  - Display the phone dialer with the person filled in

• **ACTION_VIEW** `tel:123`
  - Display the phone dialer with the given number filled in
  - VIEW action does what is considered the most reasonable thing for a particular URI

• **ACTION_EDIT** `content://contacts/people/1`
  - Edit information about the person whose identifier is "1"

https://developer.android.com/guide/components/intents-common
Intent Object - continued

- **Category**: the kind of component that should handle the intent
- Often not required, can specify multiple
  - CATEGORY_BROWSABLE - The target activity can safely be invoked by the browser
  - CATEGORY_LAUNCHER - The activity is the initial activity of a task and is listed in the system's application launcher.
  - ACTION_MAIN with category CATEGORY_LAUNCHER
    - Common Default Pair for Apps

https://developer.android.com/reference/android/content/Intent.html#standard-categories
Intent Object - continued

- **Extras** – Key-value pairs of additional information
  - putExtra() methods
- **Flags** – further metadata for system on how to handle intent.
  - Used in lower-level Intent handling.
  - E.g. for startActivity() you can set
    - `FLAG_ACTIVITY_NO_HISTORY`
    - `FLAG_ACTIVITY_NEW_TASK`
    - `FLAG_ACTIVITY_NO_ANIMATION`

https://developer.android.com/reference/android/content/Intent.html#flags
Intent intent = new Intent(Intent.ACTION_SEND);
intent.setType("plain/text");
intent.putExtra(Intent.EXTRA_EMAIL, emailAddressList);
intent.putExtra(Intent.EXTRA_SUBJECT, emailSubject);
intent.putExtra(Intent.EXTRA_TEXT, emailText);
startActivity(intent);
Intent Filters

• We looked at how to specify an Intent to launch a component

• How to tell the system that our application can handle certain intents?

Intent filters

• Specifies the type of intents that the component would like to receive
• Declared in the app’s manifest
• if you do not declare any intent filters for an activity, it can be started only with an explicit intent

<activity android:name=".BasicActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
Intent resolution

• Android puts together a list of Intent Filters
• Filters that do not match the action or category are removed from the list
• Each part of the Intent’s data URI is compared to the Intent Filters data tag
• If more than one component is resolved, they are offered to the user to select

Context

- When working with Intents, we have been using context by referring to the current activity (*this*):

  ```java
  Intent(*this*, OtherActivity::class.java)
  ```

- **Context**- An *abstract* interface to the application environment & system
  - The Context object returned can vary, depending on where you access it (e.g. from an Activity vs from a Service)

- **We need to context to**
  - invoke other components of our app
  - Invoke system services
    - `context.getSystemService(Context.WIFI_SERVICE)`
  - Access the resources tied to our app/the system

https://developer.android.com/reference/android/content/Context
Getting a reference of Context

• Different ways to get the Context:
  • `getContext();`
  • `getApplicationContext();`
  • `getBaseContext();`
  • `this`-keyword inside an Activity or Service
Working with Context

• The different Context types can be roughly divided into 2:
  • UI-based context, includes theming-related aspects
    • (e.g. using Activity as Context), `getContext()` in Fragment
  • Non-UI Context
    • `getApplicationContext()` (in Activity, Service, etc)

• (UI-based) context is meant to be short-lived
  • So don’t store UI-based context in a static variable, for example!
• Use the Context available to you from the enclosing component you’re working within.
  • Let the component lifecycle take care of releasing the Context object
  • If you need the Context reference to outlive the component, use `getApplicationContext()` instead
That’s all for today

• This week in lab:
  • We try out Navigation with Fragments and starting Activities

• Next Lecture:
  • Dynamic list handling
  • Continuing with the main Component types
    • Broadcast Receiver
    • Content Provider