Platform-Based Development: Android Programming - UI

BS UNI studies, Spring 2019/2020

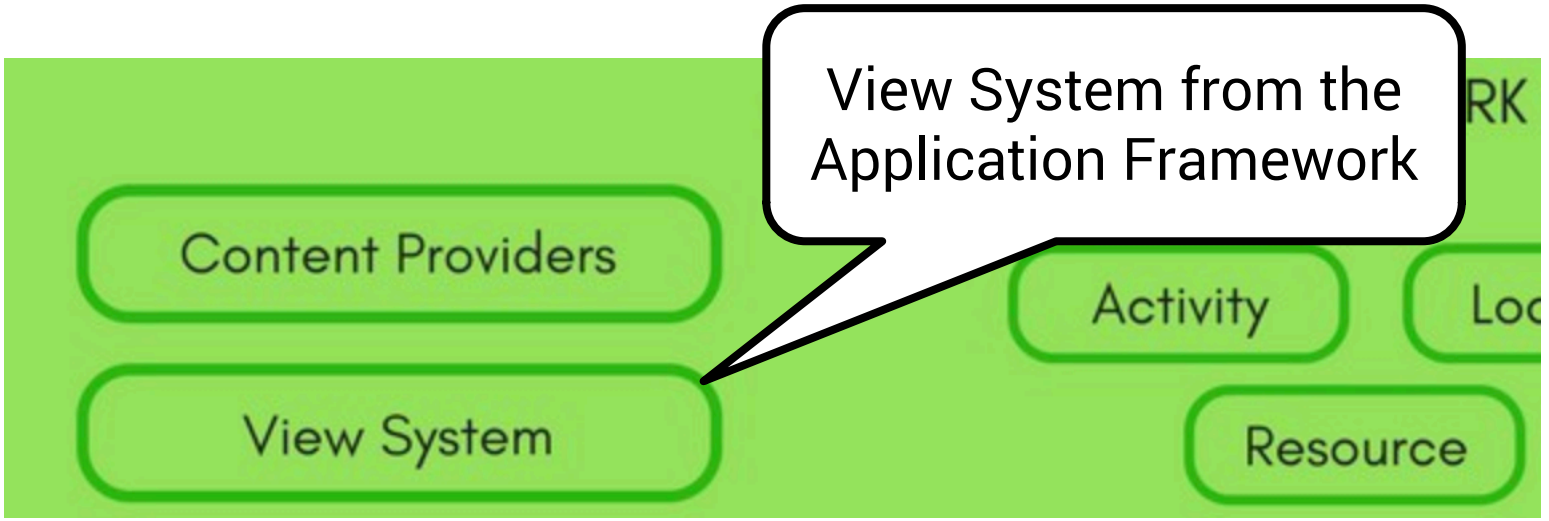
Dr Veljko Pejović
Veljko.Pejovic@fri.uni-lj.si

University *of Ljubljana*
Faculty *of Computer and Information Science*

Partly based on “Programming Handheld Systems”, Adam Porter, University of Maryland

Android UI

- UI is usually provided via Activities and Fragments
  - `setContentView(View v)`



View

- View is the main class on which the View System operates
- Responsible for drawing itself and handling events
- From View we derive other UI objects: Button, EditText fields, MapView, RadioButton, WebView, LinearLayout, etc.
- View and its derived classes can be instantiated:
  - through the XML code in Resources
  - through Java source code

View – Example

- Button:

In an XML layout:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/reg_button_text"
    android:id="@+id/register_button"
    android:layout_gravity="center_horizontal”
/>
```

The benefit of using XML – separate functionality from presentation (think MVC)

View – Example

- Button:

In Activity through Java code :

```
final Button button = (Button)
findViewById(R.id.register_button);
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        // Do something, i.e. register a user
    }
});
```

Yet, accessing through Java brings a lot of flexibility

Other View Examples

- TextView
- EditText
- CheckBox
- SeekBar
- Switch
- CalendarView

Other View Examples

- AdapterViews
  - Separation between the data (model) and children views
  - Adapter manages the data and provides it to the views
  - Example: ListView
    - Displays a scrollable list of items, where the list is populated through an Adapter
    - ArrayAdapter is the most common, but you can implement your own Adapter as well
    - Convenience classes: ListActivity and ListFragment

View Events

- Stem from user interaction and lifecycle changes
- Listeners – used for handling View events
  - `OnClickListener.onClick()` capture View clicked event
  - `OnLongClickListener.onLongClick()` capture View clicked and held for some time
  - `OnTouchListener.onTouch()` capture View touched
  - And quite a

Capturing gestures:
`GestureDetector.OnGestureListener`

View Events

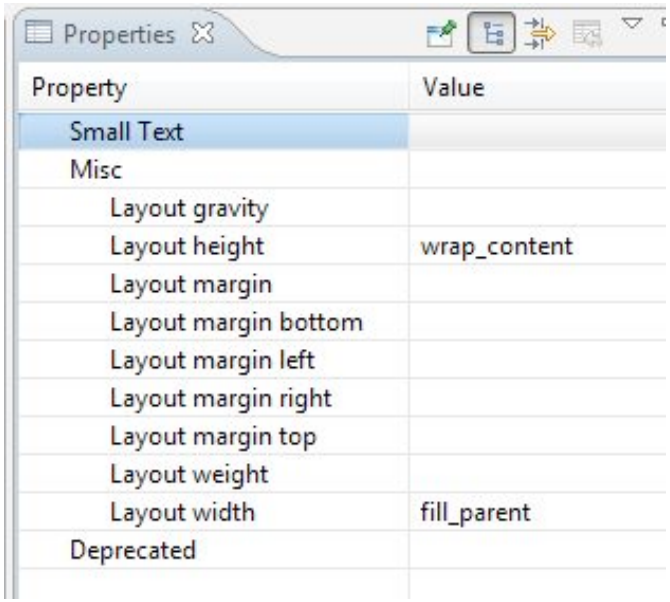
- Example – Capture `onClick` event

```
private OnClickListener mListener = new OnClickListener() {
    public void onClick(View v) {
      // do something when the button is clicked
    }
};

protected void onCreate(Bundle savedValues) {
    ...
    // Capture our button from layout
    Button button = (Button)findViewById(R.id.button);
    // Register the onClick listener with the implementation above
    button.setOnClickListener(mListener);
    ...
}
```

View Parameters

- XML attributes that define positioning, behaviour, rendering
  - Also impact children views
- Positioning and sizing should be relative in order to support different screen sizes
  - density-independent pixels (dp)
  - *wrap_content* and *match_parent*



View Parameters

- Example
  - Layout: WRAP_CONTENT vs MATCH_PARENT

```
android:layout_width="wrap_content”
android:layout_height="wrap_content”
android:text="@string/reg_button_text”
android:id="@+id/register_button”
android:layout_gravity="center_horizontal”
```

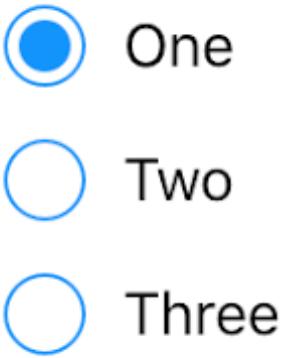
Also accessible through Java: change visibility, modify text, get checked state, etc.

Displaying Views

- Views are organised in a tree with the root View containing all the others
  - Layout Inspector in Android Studio allows you to examine the hierarchy of your views
- Displaying Views includes
  - Measuring each of the elements (calling `onMeasure()`)
  - Aligning the children (calling `onLayout()`)
  - Rendering the view (`onDraw()`)

Grouping Views

- ViewGroup
  - An invisible View that contains other Views
  - Used for grouping and organizing views
  - Example: RadioGroup, TimePicker, Spinner
- Layout
  - A ViewGroup that defines a structure for the Views it contains



Layouts

- LinearLayout
  - Child views arranged in a single horizontal or vertical row
- RelativeLayout
  - Child views are positioned relative to each other and to parent view
- ConstraintLayout
  - The latest, fastest layout that allows complex flat organisation
- But also other layouts: TableLayout, FrameLayout, GridLayout, TabLayout, etc.

App Bar

- Allows quick access to frequently used operations
  - Title, logo, navigation, action menu

- Similar to the application bar in desktop apps
- Toolbar - Introduced in API 21 (Lollipop) to replace ActionBar introduced in API 11
  - Toolbar is more general than ActionBar (which is closely tied to an Activity), as it can be placed anywhere within the view hierarchy
- Some themes come with their own bars

Use Toolbar as App Bar

```
dependencies {
  implementation 'androidx.appcompat:appcompat:1.1.0'}
```

Gradle

```
< androidx.appcompat.widget.Toolbar
    android:id="@+id/toolbar"
    android:layout_width=”match_parent"
    android:layout_height="wrap_content”
    …
/>
```

Layout XML

```
import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.Toolbar;

public class MyActivity extends AppCompatActivity{
   @Override protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_my);
       Toolbar toolbar = findViewById(R.id.toolbar);
       setSupportActionBar(toolbar);
}
```

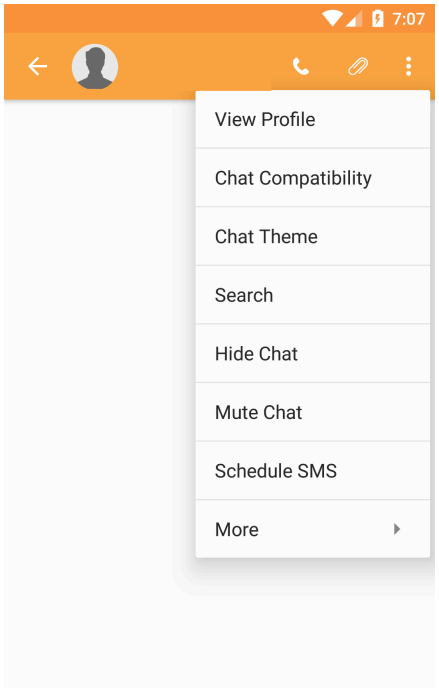
Activity Java

Menus

- Formalised in Android to provide consistent user experience across applications
- Activities support menus:
  - Add items to a menu
  - Handle `onClick` events for the menu
- Menu types:
  - Options menu
  - Context menu
  - Popup menu

Options Menu

- Primary menu where actions such as “compose new email”, “settings” and similar are added
- Define in an XML file, put in res/menu




```
<?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>
```

Options Menu

- Inflate in Activity - `onCreateOptionsMenu`

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.game_menu, menu);
    return true;
}
```

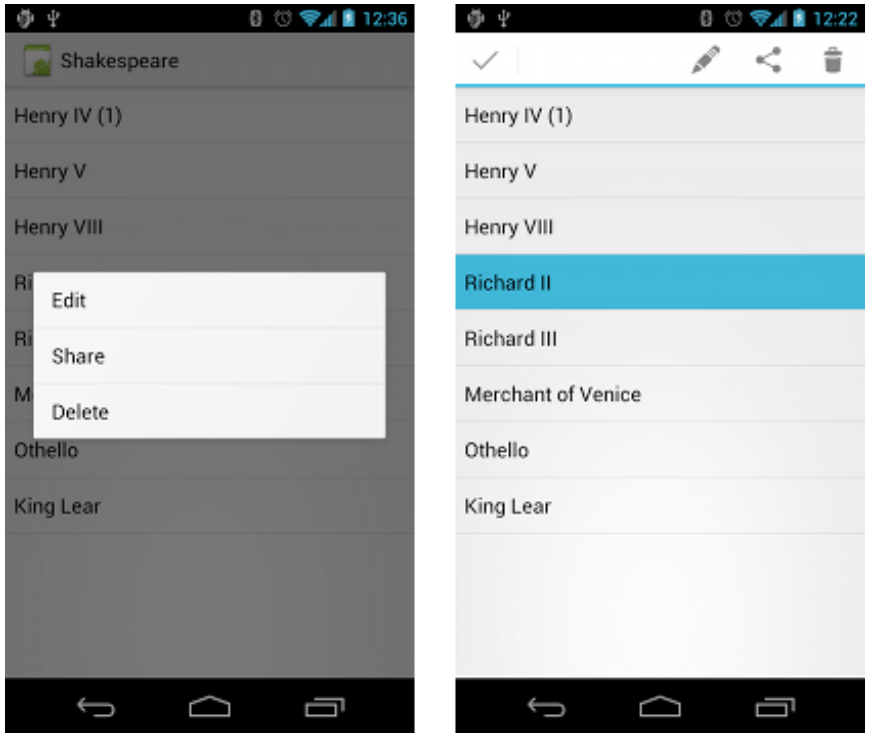
Note: Fragments have their own onCreateOptionsMenu

- Handle `onClick` events

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.new_game:
            newGame();
            return true;
        case R.id.help:
            showHelp();
            return true;
...
```

Context Menu

- Options that affect the selected item in the context of the current UI frame
- Floating context menu
- Contextual action mode



Context Menu

- Define your menu in res/menu as an XML file
- Register a View for a context menu

```
Button btn = (Button) findViewById(R.id.btn);
registerForContextMenu(btn);
```

- Inflate context menu

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
                                ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.context_menu, menu);}
```

- Capture on item selected events

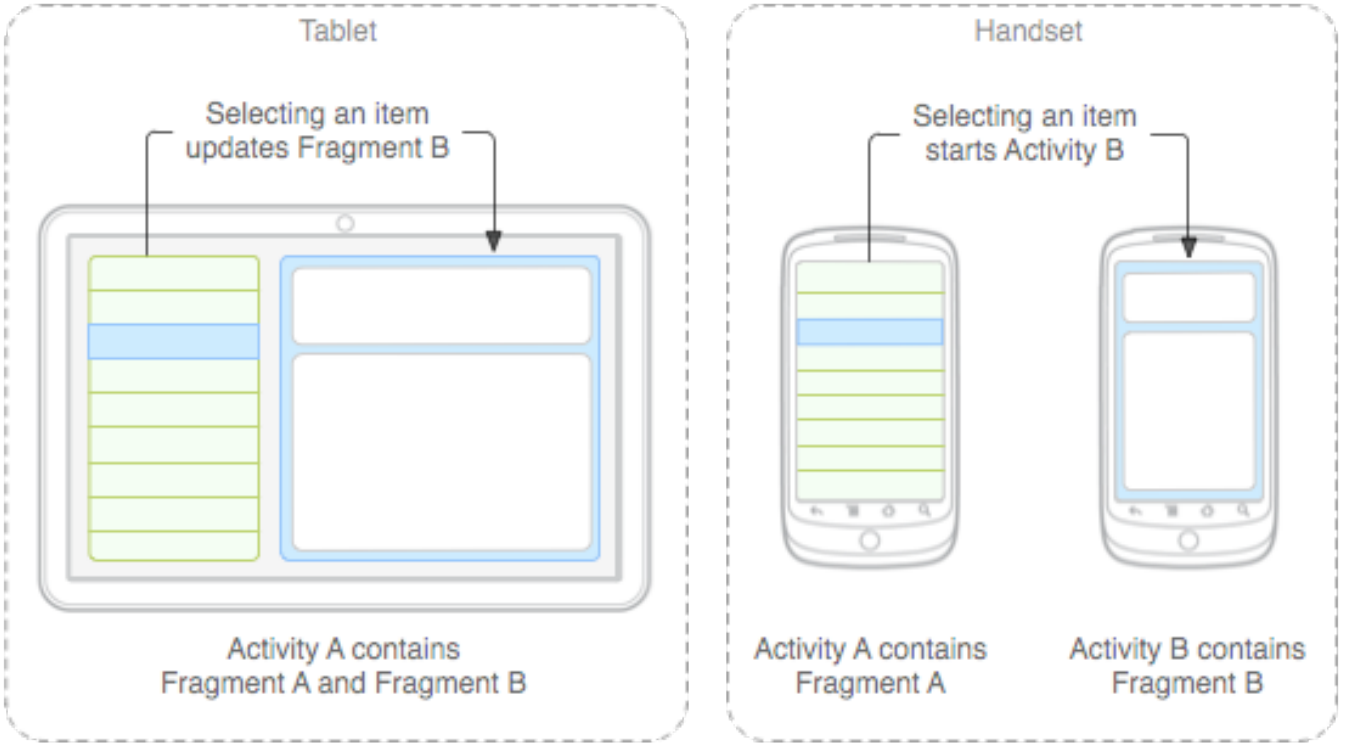
```
public boolean onContextItemSelected(MenuItem item) …
```

Fragments

- Main purpose:
  - Support more dynamic and flexible UI designs
  - Support UI element reusability
- Just like an Activity, a Fragment has
  - Layout
  - Lifecycle
- Unlike an Activity, a Fragment:
  - Needs a parent Activity
  - Need not be the only Fragment on the screen
  - Does not talk directly to other Fragments

Example Usages

- Handling devices of different screen sizes:
  - Take over UI management from the underlying Activity

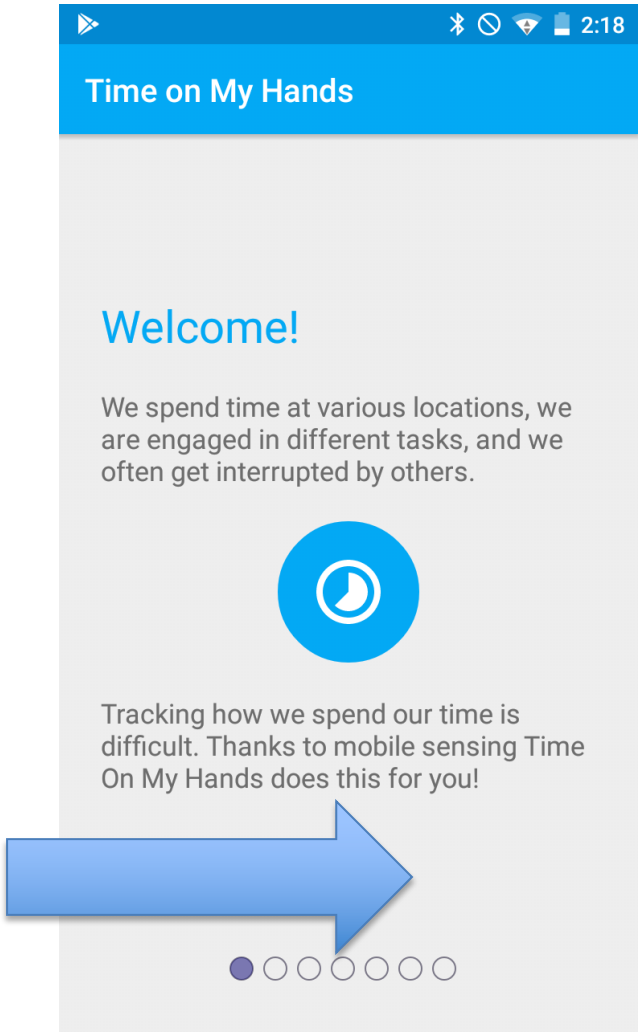


Example Usages

- Passing information among different screens:
  - Fragments are tied to the common Context, i.e. Activity
  - Storing information within the common context allows simple information sharing

Example Usages

- Flexible and advanced UI metaphors
  - Tabs
  - Swipe navigation

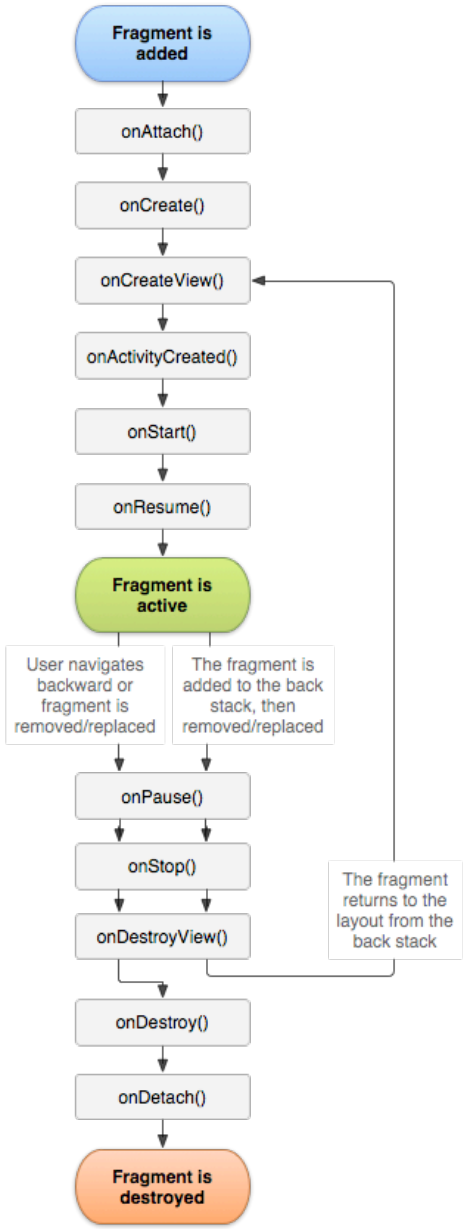


Creating a Fragment

- Create a class that extends Fragment
  - Or that extends one of the subclasses: DialogFragment, ListFragment, PreferenceFragmentCompat
- Override and implement Fragment lifecycle callbacks

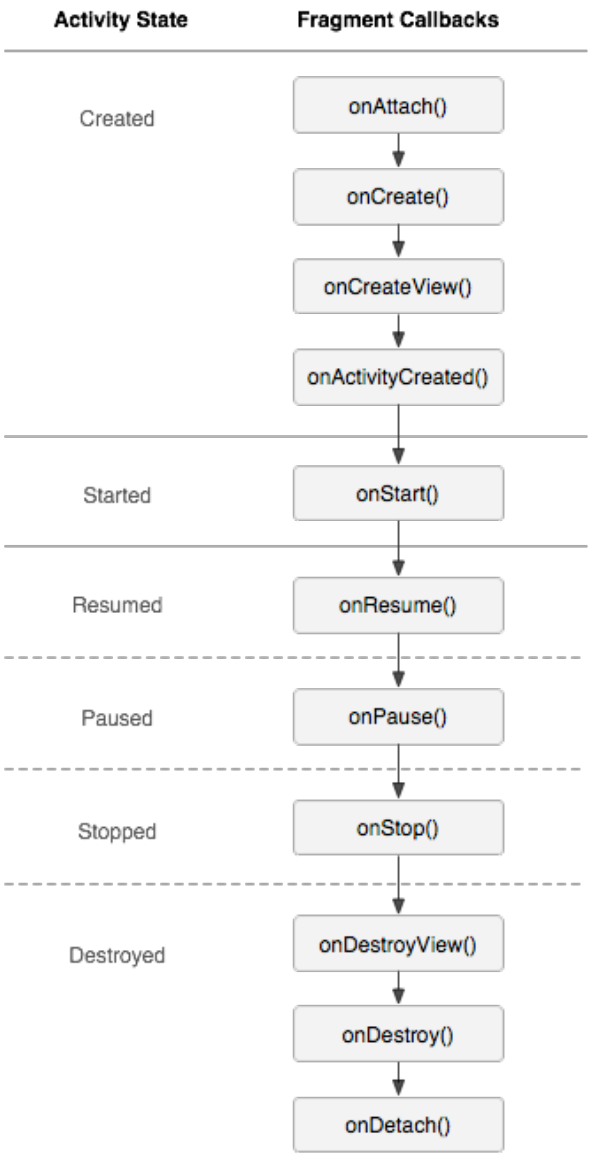
Fragment Lifecycle

- Similar to Activity lifecycle
- You should handle
  - Fragment creation in onCreate()
  - Fragment UI in onCreateView()
  - Fragment moving to the background in onPause()
  - Perhaps other calls as well



Fragment Lifecycle

- Fragment lifecycle is attached to the underlying Activity lifecycle
- Note: a Fragment is not automatically added to the backstack, but you can add it manually
  - addToBackStack()



Associating Fragment to Activity

- In an Activity's XML layout file:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <fragment android:name="com.example.news.ArticleListFragment"
            android:id="@+id/list"
            android:layout_weight="1"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
    <fragment android:name="com.example.news.ArticleReaderFragment"
            android:id="@+id/viewer"
            android:layout_weight="2"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
</LinearLayout>
```

Associating Fragment to Activity

- Or programmatically in Java using FragmentManager:

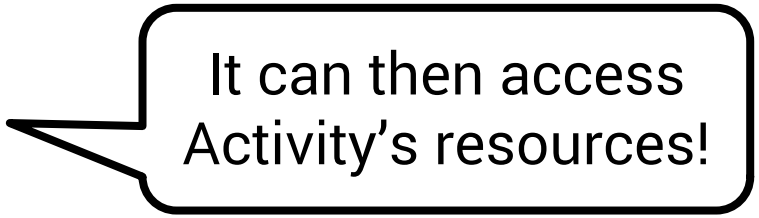
```
FragmentManager fragmentManager = getSupportFragmentManager();
FragmentTransaction fragmentTransaction =
                                    fragmentManager.beginTransaction();

ExampleFragment fragment = new ExampleFragment();
fragmentTransaction.add(R.id.fragment_container, fragment);
fragmentTransaction.commit();
```

fragment_container is a ViewGroup where the fragment screen will be shown (often FrameLayout)

Communicating between Fragment and Activity

- Fragment can access the underlying Context (Activity)
  - getContext() or getActivity()



- Activity can access fragments via FragmentManager
  - findFragmentByID() or findFragmentByTag()

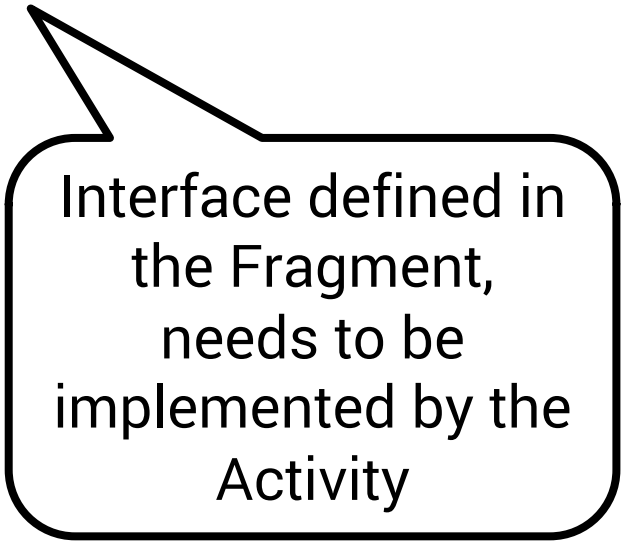
```
ExampleFragment fragment =
                (ExampleFragment) getSupportFragmentManager()
                     .findFragmentById(R.id.example_fragment);
```

Communicating between Fragment and Activity

- Listeners
  - Fragment A has an interface that the underlying Activity implements
  - The interface contains a method that is called when the monitored data is modified
    - The Activity can now record/process this data
    - The Activity can use this data in another Fragment

Listener Communication Example

```
public static class FragmentA extends ListFragment {
    ...
    // Container Activity must implement this interface
    public interface OnArticleSelectedListener {
        public void onArticleSelected(Uri articleUri);
    }
    ...
}
```



Listener Communication Example

```
public static class FragmentA extends ListFragment {
    OnArticleSelectedListener listener;
    ...
    @Override
    public void onAttach(Context context) {
        super.onAttach(context);
        try {
            listener = (OnArticleSelectedListener) context;
        } catch (ClassCastException e) {
            throw new ClassCastException(context.toString() +
" must implement OnArticleSelectedListener");
        }
    }
    ...
}
```

The Fragment “refuses” an Activity that does not implement the Interface

Listener Communication Example

```
public static class FragmentA extends ListFragment {
    OnArticleSelectedListener listener;
    ...
    @Override
    public void onListItemClick(ListView l, View v,
                                int position, long id) {
       // Append the clicked item's row ID with
       //the content provider Uri
        Uri noteUri = ContentUris
             .withAppendedId(ArticleColumns.CONTENT_URI, id);
        // Send the event and Uri to the host activity
        listener.onArticleSelected(noteUri);
    }
    ...
}
```

The Fragment sets the data value for the Activity

Fragment vs Support Fragment

- Android Support Library
  - Created to provide backwards compatibility and enable new features on old devices
  - Became mainstream
- Three options for Fragments:
  - android.app.Fragment
  - android.support.v4.app.Fragment
  - androidx.fragment.app.Fragment
- FragmentActivity
  - Support for work with Fragments

Currently maintained/ prefer this one!

AppCompatActivity extends it!

Android Architecture Components

- A set of components designed to make common tasks quicker and easier to do:
  - Persistence and data sharing among Fragments (ViewModel)
  - Displaying data from a database (LiveData)
  - Make classes aware of the Activity lifecycle (LifecycleObserver)
  - …and a few more things
- Imported from androidx.* package
- Will be covered in a separate lecture