CS360 Lab 3 Presentation

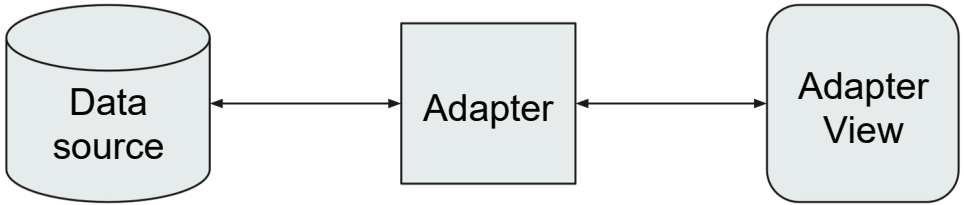
Spring 2026

Overview

- Custom ArrayAdapter
- Fragments
- Java conventions
- Lab demo

Adapter and AdapterView

- The content for our layout is dynamic or not predetermined, so we have to render it at runtime
- the **Adapter** retrieves the data (from a source such as an array or a database query) and converts each entry into a view that can be added into the AdapterView layout.



Customize ArrayAdapter

To customize the appearance of each item you can create a view for each item that's something other than a **TextView** (for example, if you want an **ImageView** for each array item), extend the **ArrayAdapter** class and override **getView()** to return the type of view you want for each item

Source:
https://developer.android.com/develop/ui/views/layout/declaring-layout#FillingTheLayout

Override getView()

The **getView()** method is automatically called when the list item view is ready to be displayed or about to be displayed.

It should return the View that displays the data at the specified position in the data set.

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    // Get the data item for this position
    User user = getItem(position);
    // Check if an existing view is being reused, otherwise inflate the view
    View view;
    if (convertView == null) {
        view = LayoutInflater.from(getContext()).inflate(R.layout.list_view,
parent, false);
    } else {
        view = convertView;
    }
    // Lookup view for data population
    TextView name = view.findViewById(R.id.name);
    ImageView avatar = view.findViewById(R.id.avatar);
    // Populate the data into the template view using the data object
    name.setText(user.name);
    avatar.setImageResouces(user.avatar);
    // Return the completed view to render on screen
    return view;
}
```

Fragments

- Represents a reusable portion of your app's UI.
- Defines and manages its own layout, has its own lifecycle, and can handle its own input events.
- Cannot live on their own – they must be hosted by an activity or another fragment

Source:
https://developer.android.com/guide/fragments

Fragment Lifecycle

1)	onAttach(Activity)	it is called only once when it is attached with activity.
2)	onCreate(Bundle)	It is used to initialize the fragment.
3)	onCreateView(LayoutInflater, ViewGroup, Bundle)	creates and returns view hierarchy.
4)	onActivityCreated(Bundle)	It is invoked after the completion of onCreate() method.
5)	onViewStateRestored(Bundle)	It provides information to the fragment that all the saved state of fragment view hierarchy has been restored.
6)	onStart()	makes the fragment visible.
7)	onResume()	makes the fragment interactive.
8)	onPause()	is called when fragment is no longer interactive.
9)	onStop()	is called when fragment is no longer visible.
10)	onDestroyView()	allows the fragment to clean up resources.
11)	onDestroy()	allows the fragment to do final clean up of fragment state.
12)	onDetach()	It is called immediately prior to the fragment no longer being associated with its

Source: https://stackoverflow.com/questions/39418249/meaning-of-fragment-having-its-own-lifecycle

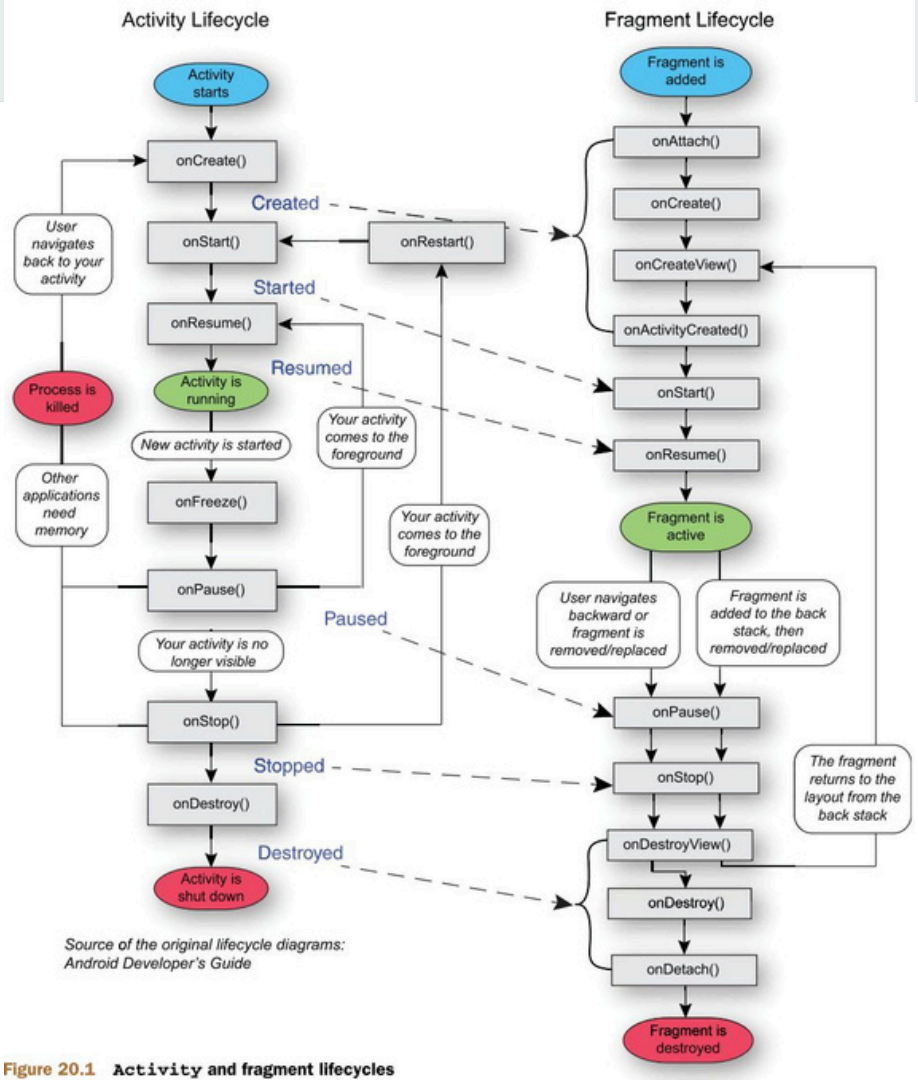
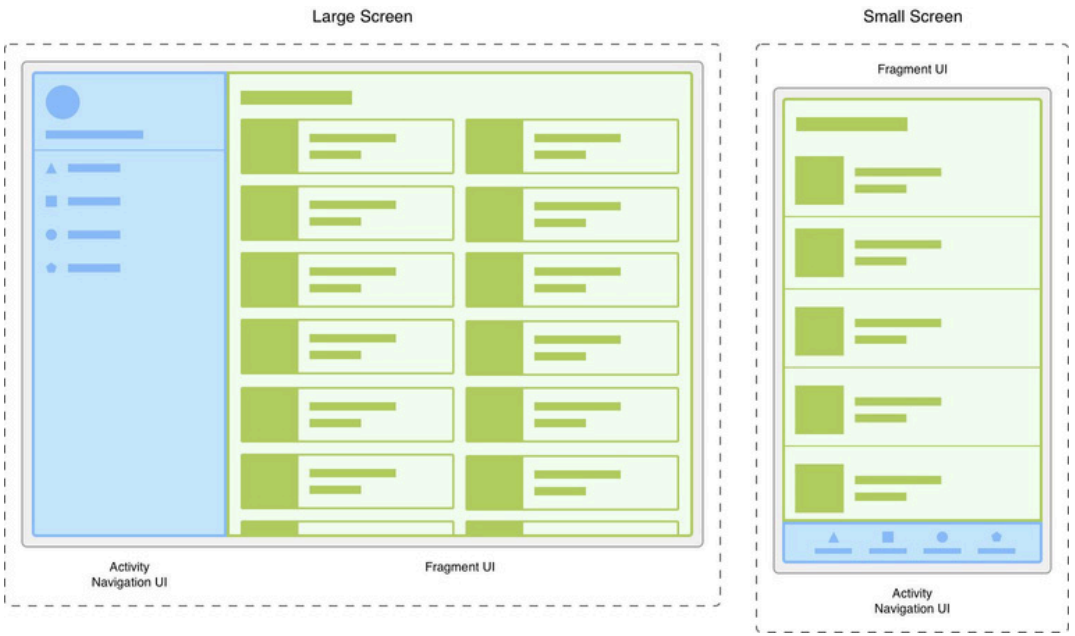


Figure 20.1 Activity and fragment lifecycles

Fragments is not required, but nice to have. It could speed up your application and save resources.



Java Code Conventions

Code conventions are important to programmers for a number of reasons:

- 80% of the lifetime cost of a piece of software goes to maintenance.
- Hardly any software is maintained for its whole life by the original author.
- Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.
- If you ship your source code as a product, you need to make sure it is as well packaged and clean as any other product you create.

Source:
https://www.oracle.com/java/technologies/javase/codeconventions-introduction.html