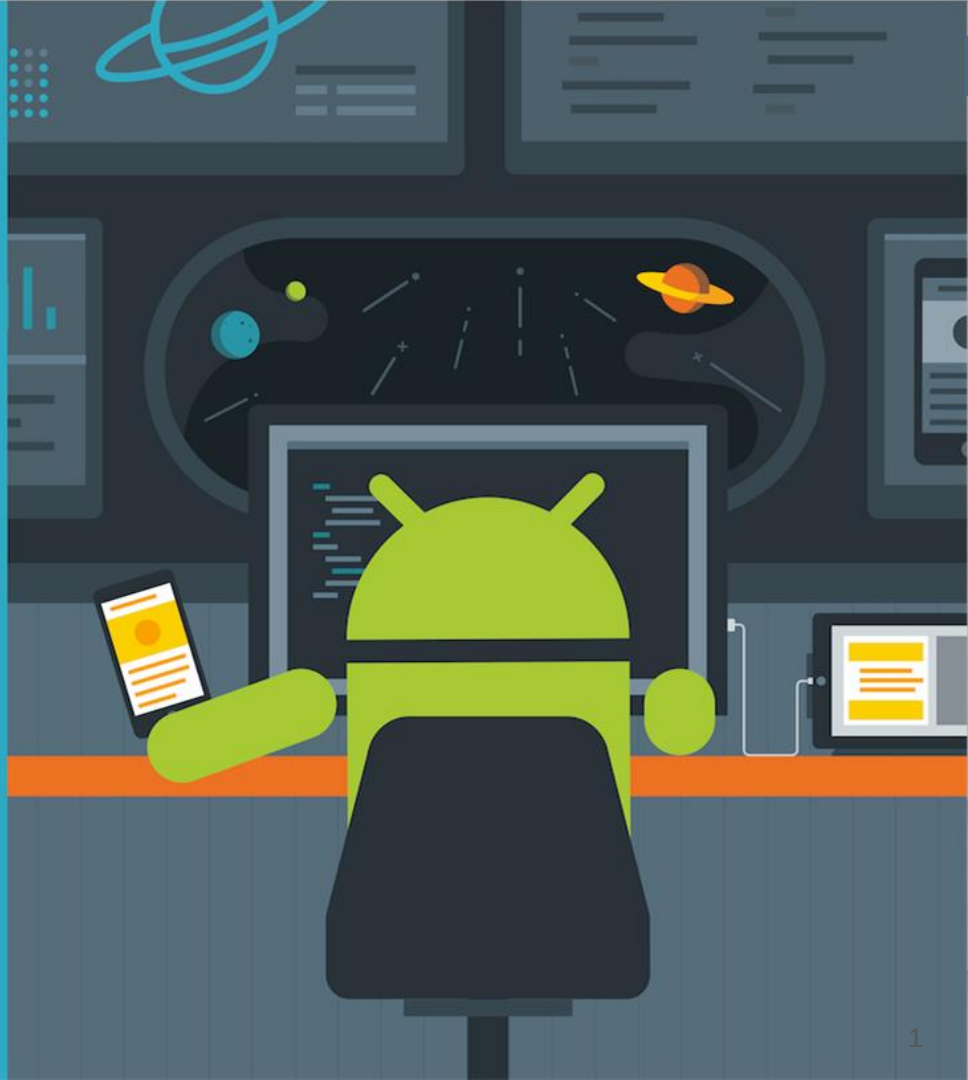
Advanced Android Development

Fragments

Lesson 1



1.1 Fragments

A reusable UI component with its own lifecycle

Contents

- Understanding the `Fragment` class
- Creating a `Fragment`
- Using a layout for a `Fragment`
- Adding a `Fragment` to an `Activity`

Understanding the Fragment class

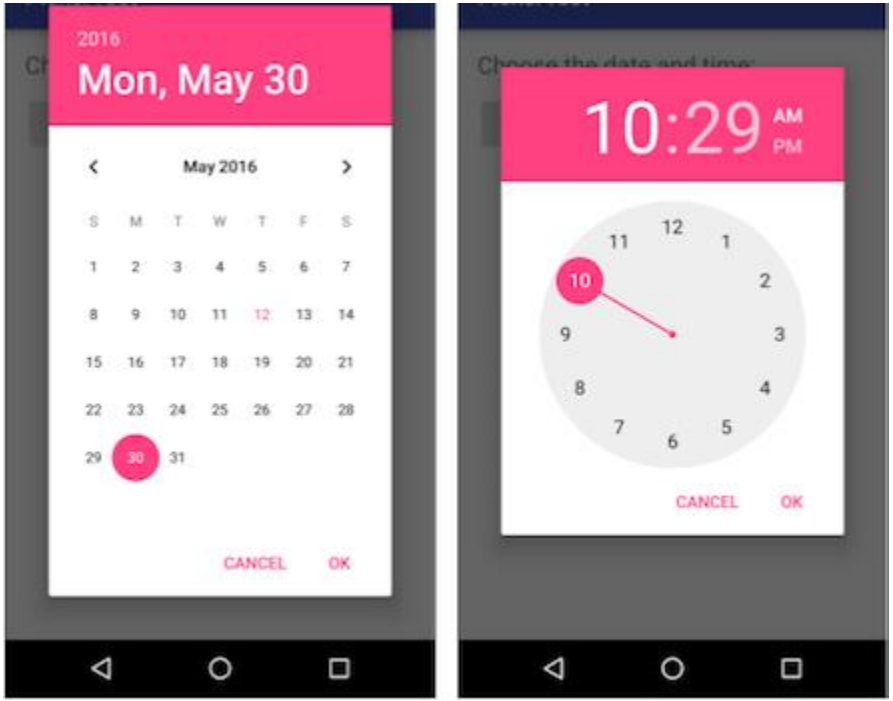
Fragment class

- Contains a portion of the UI and its behavior
- Has its own lifecycle states (like an `Activity`)
- Reusable—share across multiple activities
  - Each `Fragment` instance exclusively tied to host `Activity`
  - `Fragment` code defines layout and behavior
- Represents sections of a UI for different layouts

Example: Pickers use DialogFragment

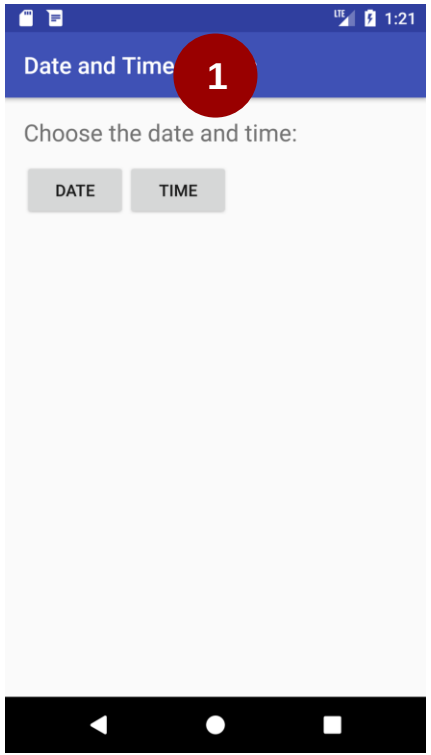
Date and time pickers:

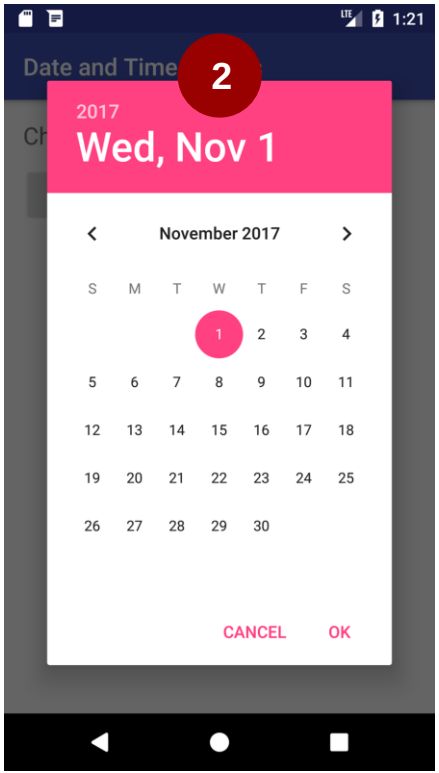
Extend `DialogFragment` (`Fragment` subclass)



DialogFragment hosted by Activity

1. `Activity` before adding date picker fragment
2. Date picker fragment appears on top of `Activity`





Add a Fragment to an Activity

- Static part of UI (in `Activity` layout):

  on screen during entire `Activity` lifecycle

- Dynamic part of UI:

  added and removed while `Activity` is running

Benefits of using fragments

- Reuse a `Fragment` in more than one `Activity`
- Add or remove dynamically as needed
- Integrate a mini-UI within an `Activity`
- Retain data instances after a configuration change
- Represent sections of a layout for different screen sizes

Steps to using a fragment

1. Create a subclass of `Fragment`
2. Create a layout for the `Fragment`
3. Add `Fragment` to a host `Activity`
    - Statically in layout
    - Dynamically using fragment transactions

Creating a Fragment

Add new Fragment in Android Studio

- Expand **app > java** in project and select package name
- Choose **File > New > Fragment > Fragment (Blank)**
- Check the **Create layout XML** option for layout
- Other options:
  - **Include fragment factory methods** to include a `newInstance()` method to instantiate the `Fragment`
  - **Include interface callbacks** to define an interface with callbacks

New Fragment

```
public class SimpleFragment extends Fragment {
    public SimpleFragment() {
        // Required empty public constructor
    }
    ...
}
```

Extend the Fragment class

- Extend `Fragment` class
  - `public class SimpleFragment extends Fragment`
- Extend specific `Fragment` subclass:
  - `DialogFragment`: Floating dialog (examples: date and time pickers)
  - `ListFragment`: List of items managed by adapter
  - `PreferenceFragment`: Hierarchy of `Preference` objects (useful for Settings)

Using a layout for a Fragment

Create a layout for a Fragment

- **Create layout XML** option adds XML layout
- `Fragment` callback `onCreateView()` creates `View`
  - Override this to inflate the `Fragment` layout
  - Return `View`: root of `Fragment` layout

Inflate the layout for the Fragment (1)

```
@Override
public View onCreateView(LayoutInflater inflater,
            ViewGroup container, Bundle savedInstanceState) {
    // Inflate the fragment layout and return it as root view.
    return inflater.inflate(R.layout.fragment_simple,
                                                container, false);
}
```

Inflate the layout for the Fragment (2)

- Fragment layout is inserted into `container` `ViewGroup` in `Activity` layout
- `LayoutInflater` inflates layout and returns `View` layout root to `Activity`
- `Bundle` `savedInstanceState` saves previous `Fragment` instance

Inflate the layout for the Fragment (3)

```
return inflater.inflate(R.layout.fragment_simple, container, false);
```

- Resource ID of layout (`R.layout.fragment_simple`)
- `ViewGroup` to be parent of inflated layout (`container`)
- `Boolean`: Whether layout should be attached to parent
  - Should be `false`
  - If adding `Fragment` in code, don't pass `true` (creates a redundant `ViewGroup`)

Adding a Fragment to an Activity

Add a Fragment to an Activity

- Add ***statically*** in `Activity` layout, visible for entire `Activity` lifecycle
- Add (or remove) ***dynamically*** as needed during `Activity` lifecycle using `Fragment` transactions

Add a Fragment statically

1. Declare `Fragment` inside `Activity` layout (`activity_main.xml`) using `<fragment>` tag
2. Specify layout properties for the `Fragment` as if it were a `View`

Static Fragment example

Adding `SimpleFragment` to `Activity` layout:

```
<fragment android:name="com.example.appname.SimpleFragment"
          android:id="@+id/simple_fragment"
          android:layout_weight="2"
          android:layout_width="0dp"
          android:layout_height="match_parent" />
```

Add a Fragment dynamically

1. Specify `ViewGroup` for `Fragment` in layout
2. Instantiate the `Fragment` in `Activity`
3. Instantiate `FragmentManager`
    - Use `getSupportFragmentManager()` for compatibility
4. Use `Fragment` transactions

Specify a ViewGroup for the Fragment

Specify `ViewGroup` to place `Fragment` (such as `FrameLayout`):

```
<FrameLayout
    android:id="@+id/fragment_container"
    android:name="SimpleFragment"
    tools:layout="@layout/fragment_simple"
    ... />
```

Instantiate the Fragment

1. Create `newInstance()` factory method in `Fragment`:

```
public static SimpleFragment newInstance() {
        return new SimpleFragment();
}
```

1. In `Activity`, instantiate `Fragment` by calling `newInstance()`:

```
SimpleFragment fragment = SimpleFragment.newInstance();
```

Instantiate FragmentManager

In `Activity`, get instance of `FragmentManager` with `getSupportFragmentManager()`:

```
FragmentManager fragmentManager = getSupportFragmentManager();
```

Use the Support Library version—`getSupportFragmentManager()` rather than `getFragmentManager()`—for compatibility with earlier Android versions

Use Fragment transactions

`Fragment` operations are wrapped into a ***transaction***:

- Start transaction with `beginTransaction()`
- Do all `Fragment` operations (add, remove, etc.)
- End transaction with `commit()`

Fragment transaction operations

- Add a `Fragment` using `add()`
- Remove a `Fragment` using `remove()`
- Replace a `Fragment` with another using `replace()`
- Hide and show a `Fragment` using `hide()` and `show()`
- Add `Fragment` transaction to back stack using `addToBackStack(null)`

Fragment transaction example (1)

```
FragmentTransaction fragmentTransaction =
            fragmentManager.beginTransaction();
fragmentTransaction
            .add(R.id.fragment_container, fragment)
            .addToBackStack(null)
            .commit();
```

Fragment transaction example (2)

In the code on the previous slide:

- `add()` arguments:
  - The `ViewGroup` (`fragment_container`)
  - The `fragment` to add
- `addToBackStack(null)`:
  - Add transaction to back stack of `Fragment` transactions
  - Back stack managed by the `Activity`
  - User can press Back button to return to previous `Fragment` state

Fragment transaction example (3)

Removing a `Fragment`:

```
SimpleFragment fragment = (SimpleFragment) fragmentManager
                    .findFragmentById(R.id.fragment_container);
if (fragment != null) {
     FragmentTransaction fragmentTransaction =
                    fragmentManager.beginTransaction();
     fragmentTransaction.remove(fragment).commit();
}
```

What's next?

- Concept chapter: 1.1 Fragments
- Practical: 1.1 Creating a Fragment with a UI

END