

CS 213 – Software Methodology

Spring 2019

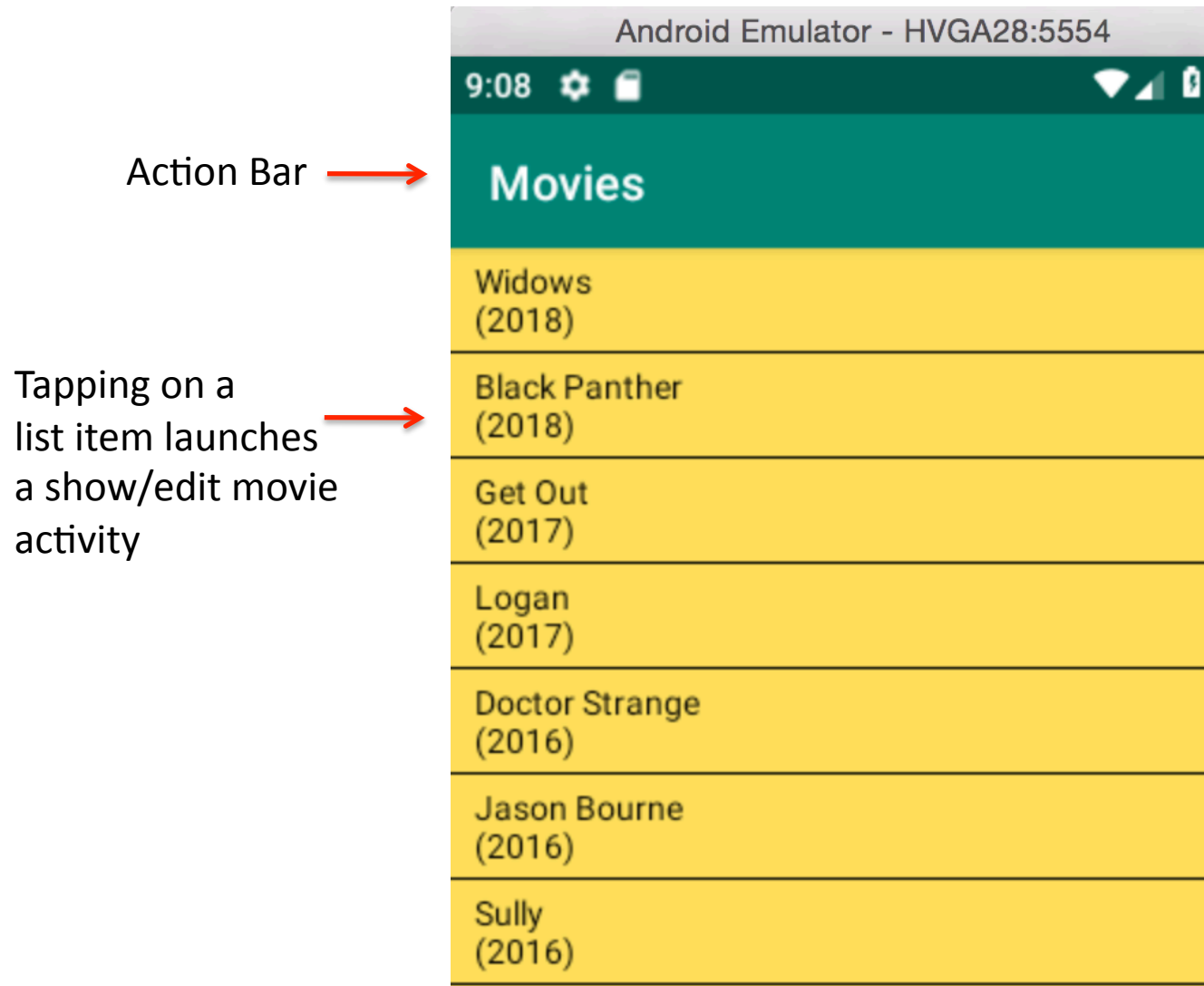
Sesh Venugopal

Lecture 21 – Apr 16
Android Programming

Activities | Dialogs | Icons | Device I/O

Movies List Project

List set up like in the Rutgers Bus Routes app



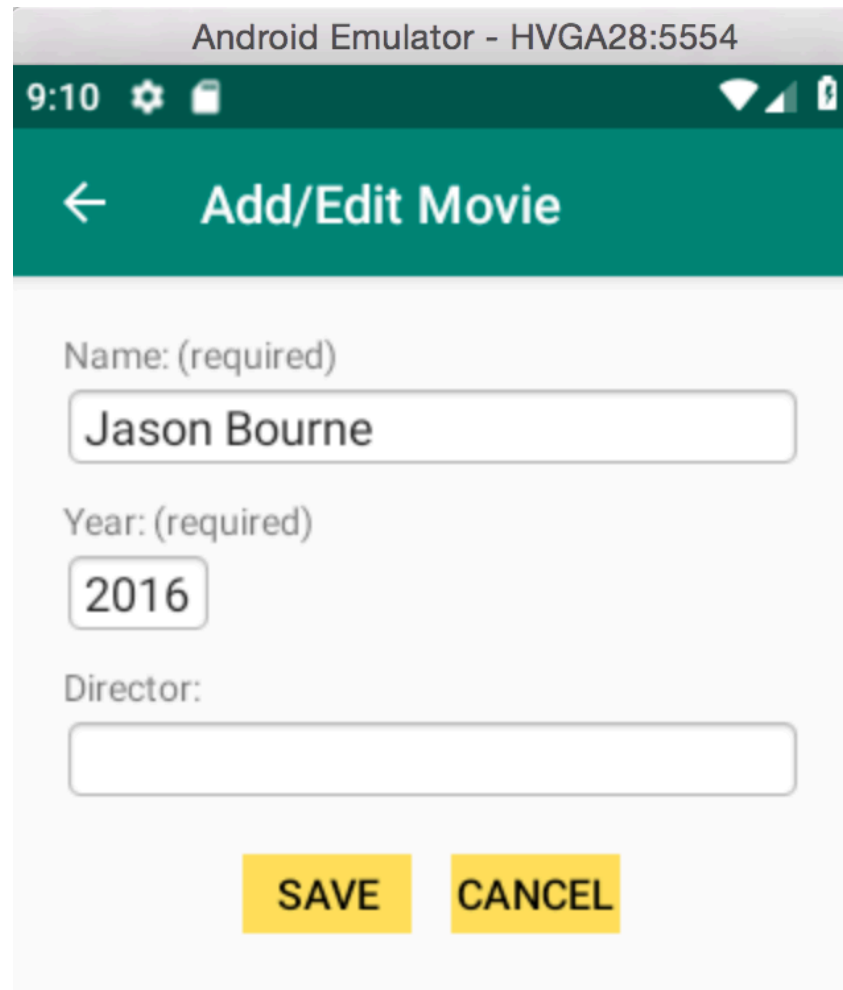
Part 1:

Launching Activity for Result

AddEditMovie Activity

This activity sets up text fields that are pre-populated if it is launched for Show/Edit

A movie item was
tapped in the parent
movie list activity



The screenshot shows an Android emulator window titled "Android Emulator - HVGA28:5554". The status bar at the top displays the time 9:10, a gear icon, a folder icon, and signal/battery indicators. The app's header is a teal bar with a back arrow and the text "Add/Edit Movie". The main form contains three fields: "Name: (required)" with the value "Jason Bourne", "Year: (required)" with the value "2016", and "Director:" with an empty field. At the bottom are two yellow buttons labeled "SAVE" and "CANCEL". Two red arrows point from the text "A movie item was tapped in the parent movie list activity" to the "Name" and "Year" fields, indicating they are pre-populated.

Android Emulator - HVGA28:5554

9:10

← Add/Edit Movie

Name: (required)

Jason Bourne

Year: (required)

2016

Director:

SAVE CANCEL

Launching Activity for Result

The Show/Add/Edit activity might return a result (when Save button is clicked) that needs to be communicated back to the launching parent

Movies.java

```
public static final int EDIT_MOVIE_CODE=1;
```

```
...  
public void showMovie(int pos) {  
    Bundle bundle = new Bundle();  
    Movie movie = movies.get(pos);  
    bundle.putInt(AddEditMovie.MOVIE_INDEX,pos);  
    bundle.putString(AddEditMovie.MOVIE_NAME,movie.name);  
    bundle.putString(AddEditMovie.MOVIE_YEAR,movie.year);  
    bundle.putString(AddEditMovie.MOVIE_DIRECTOR,movie.director);  
    Intent intent = new Intent(this, AddEditMovie.class);  
    intent.putExtras(bundle);  
    startActivityForResult(intent, EDIT_MOVIE_CODE);  
}
```

Bundle for movie info to send
to child activity for display

An activity could launch several
children activities with different
request codes



So when a child activity returns, this
code will be used to determine which
of potentially several children
activities it is

Implementing activity AddEditMovie

File -> New -> Activity -> Basic Activity

Creates a new basic activity with an app bar.

Activity Name:

Layout Name:

Title:

☐ Launcher Activity

☐ Use a Fragment

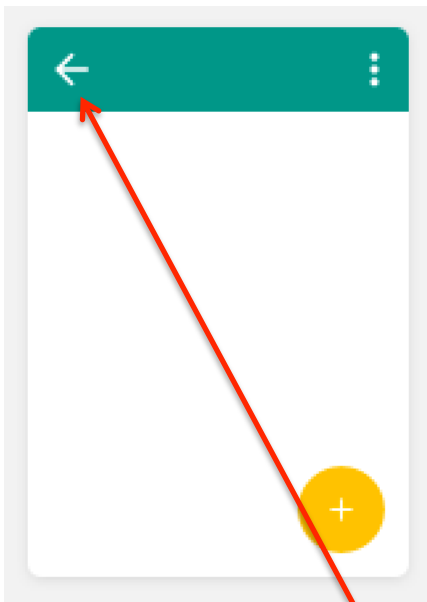
Hierarchical Parent: ▼ ...

Package name: ▼

Source Language: ▼

Also creates a `content_add_edit_moie.xml` in which you should put your actual add/edit view xml

The hierarchical parent activity, used to provide a default implementation for the 'Up' button



AddEditMovie Class

Set up with constants and instance fields:

```
public class AddEditMovie  
extends AppCompatActivity {
```

Keys used to ship info from and
to the parent **Movies** activity

```
    public static final String MOVIE_INDEX = "movieIndex";  
    public static final String MOVIE_NAME = "movieName";  
    public static final String MOVIE_YEAR = "movieYear";  
    public static final String MOVIE_DIRECTOR = "movieDirector";
```

```
    private int movieIndex; ← The position of the movie in the array list of  
                             movies in the Movies activity
```

```
    private EditText movieName, movieYear, movieDirector;
```

```
    ...
```

```
}
```

↑
Text fields in fill out form

AddEditMovie Class

If the incoming **Bundle** is not null (if called to Show/Edit), then get info and populate fields

```
public class AddEditMovie
extends AppCompatActivity {
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        // get the fields
        movieName = findViewById(R.id.movie_name);
        movieYear = findViewById(R.id.movie_year);
        movieDirector = findViewById(R.id.movie_director);

        // see if info was passed in to populate fields
        Bundle bundle = getIntent().getExtras();
        if (bundle != null) {
            movieIndex = bundle.getInt(MOVIE_INDEX);
            movieName.setText(bundle.getString(MOVIE_NAME));
            movieYear.setText(bundle.getString(MOVIE_YEAR));
            movieDirector.setText(bundle.getString(MOVIE_DIRECTOR));
        }
    }
}
```


AddEditMovie Event Handling

Save and Cancel buttons are fitted with respective event handling methods in `AddEditMovie` class

The screenshot shows a mobile application interface for adding or editing a movie. It has a teal header bar with a back arrow and the title 'Add/Edit Movie'. Below the header, there are three input fields: 'Name: (required)' with the text 'Jason Bourne', 'Year: (required)' with the text '2016', and 'Director:' which is empty. At the bottom, there are two yellow buttons labeled 'SAVE' and 'CANCEL'. Red arrows point from the text annotations below to these buttons.

In layout xml
event handler:

`android:onClick="save"`

Method name

In layout xml
event handler:

`android:onClick="cancel"`

Method name

AddEditMovie Event Handling

Handling **Cancel** event in **AddEditMovie** class

```
public void cancel(View view) {  
    setResult(RESULT_CANCELED);  
    finish();  
}
```

Button that was clicked

Calling this method
results in termination of
activity, with a return to
the previous activity on
call stack

Result code that is sent back to parent
activity, code is a constant defined in
the **Activity** class (of which
AddEditMovie is a subclass)

AddEditMovie Event Handling

Handling **Save** event in **AddEditMovie** class

```
public void save(View view) {  
    // gather all data from text fields  
    String name = movieName.getText().toString();  
    ...  
    // pop up dialog if errors in input, and return  
    ...  
    // make Bundle  
    Bundle bundle = new Bundle();  
    bundle.putInt(MOVIE_INDEX, movieIndex);  
    bundle.putString(MOVIE_NAME, name);  
    bundle.putString(MOVIE_YEAR, year);  
    bundle.putString(MOVIE_DIRECTOR, director);  
  
    // send back to caller  
    Intent intent = new Intent();  
    intent.putExtras(bundle);  
    setResult(RESULT_OK, intent);  
  
    finish(); // pops activity from the call stack, returns to parent  
}
```



Mechanism to send result
back to parent activity

AddEditMovie finishes

- When `AddEditMovie` finishes, execution returns to `Movies` via a call to its `onActivityResult` method (callback)

`@Override`

```
protected void onActivityResult(int requestCode,  
                                int resultCode,  
                                Intent intent) {
```

```
    if (resultCode != RESULT_OK) { return; }
```

```
    Bundle bundle = intent.getExtras();  
    if (bundle == null) { return; }
```

```
    // gather all info passed back by launched activity  
    String name = bundle.getString(AddEditMovie.MOVIE_NAME);  
    String year = bundle.getString(AddEditMovie.MOVIE_YEAR);  
    String director = bundle.getString(AddEditMovie.MOVIE_DIRECTOR);  
    int index = bundle.getInt(AddEditMovie.MOVIE_INDEX);
```

```
    ...
```

```
}
```

Code that was sent
in by `Movies` when
`AddEditMovie` was
launched

Code returned by
`AddEditMovie`

Intent returned by
`AddEditMovie`

Only relevant if `AddEditMovie` was called for
Show/Edit (this was sent to `AddEditMovie`, and is
passed back)

AddEditMovie finishes

- `onActivityResult` method distinguishes between return from Show/Edit and Add

```
@Override
protected void onActivityResult(int requestCode,
                                int resultCode,
                                Intent intent) {

    ...

    Movie movie = movies.get(index);
    movie.name = name;
    movie.year = year;
    movie.director = director;

    // redo the adapter to reflect change
    listView.setAdapter(
        new ArrayAdapter<Movie>(this,
                                R.layout.movie, movies));

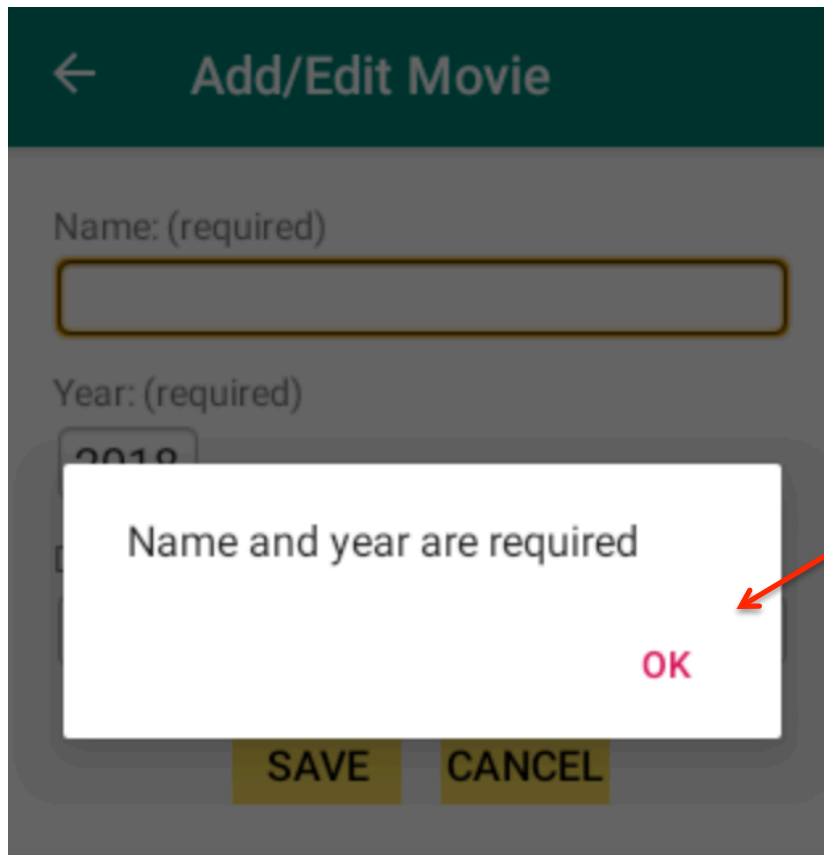
    ...
}
```

Adapter has to be redone
since source content has changed

Part 2: Dialogs

Popping up an error dialog

If either the movie name or year is missing, a dialog is popped up



Dialog is a subclass of
`DialogFragment`

Popping up an error dialog

See <https://developer.android.com/guide/topics/ui/dialogs.html>

From the package `android.support.v4.app`

```
public class MovieDialogFragment extends DialogFragment {  
    public static final String MESSAGE_KEY = "message_key";
```

The actual text to be shown can be passed in when the fragment instance is created, as value for this key

```
@Override  
public Dialog onCreateDialog(Bundle savedInstanceState) {  
    ... // create the dialog  
}  
}
```


DialogFragment onCreateDialog

The `onCreate` method of `DialogFragment` creates an `AlertDialog`, using a standard suggested coding process:

`android.support.v7.app.AlertDialog`

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    // Use the Builder class for convenient dialog construction
    Bundle bundle = getArguments();
    AlertDialog.Builder builder =
        new AlertDialog.Builder(getActivity());
    builder.setMessage(bundle.getString(MESSAGE_KEY))
        .setPositiveButton("OK", (dialog,id) -> {});
}
```

message sent in when Fragment is created

`setNegativeButton` would allow us to set up a Cancel button, which we don't need here since it's just an info dialog

```
    // Create the AlertDialog object and return it
    return builder.create();
}
```

Showing the dialog with required message

The `save` method of `AddEditMovie` checks if required fields are filled, and if not, creates and shows an instance of the `MovieDialogFragment`:

```
public void save(View view) {  
    ...  
    // name and year are mandatory  
    if (name == null || name.length() == 0 ||  
        year == null || year.length() == 0) {  
  
        Bundle bundle = new Bundle();  
        bundle.putString(MovieDialogFragment.MESSAGE_KEY,  
                        "Name and year are required");  
  
        DialogFragment newFragment = new MovieDialogFragment();  
        newFragment.setArguments(bundle);  
        newFragment.getDialog().show();  
  
        return;    // does not quit activity, just returns from method  
    }  
    ...  
}
```

Part 3: Using Icons

Using an Icon for Adding Movie

- We will use a '+' icon to add movies. This icon will show up as an item in the Action/App Bar
- There are prefab icons supplied by the Android guys for a whole lot of standard tasks, including one to add content (such as songs in our app)

Go to [Material Icons \(https://material.io/icons\)](https://material.io/icons)

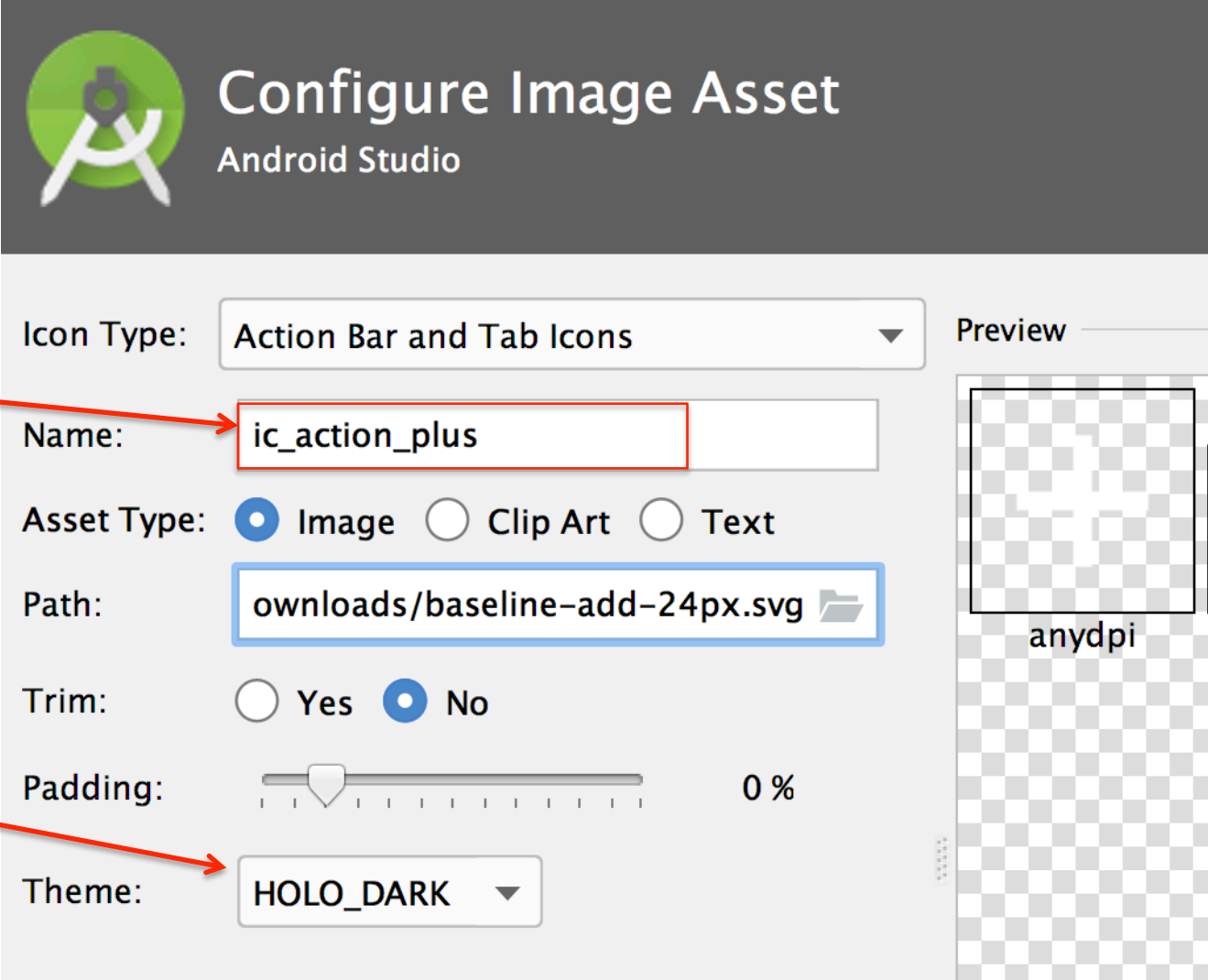
On this page, search for “Content” – this will show a collection of Content icons. The first one of them is the '+'

Click on the '+' icon: this brings up a tool bar at the bottom of the browser page where you can download the SVG 24 (24px) image [baseline_add_24px.svg](#)

Adding icons to project

Right click on [res](#), then choose [New -> Image Asset](#), then configure like this

Name that will be used in app



Configure Image Asset
Android Studio

Icon Type: Action Bar and Tab Icons

Name: **ic_action_plus**

Asset Type: ☒ Image ☐ Clip Art ☐ Text

Path: **downloads/baseline-add-24px.svg**

Trim: ☐ Yes ☒ No

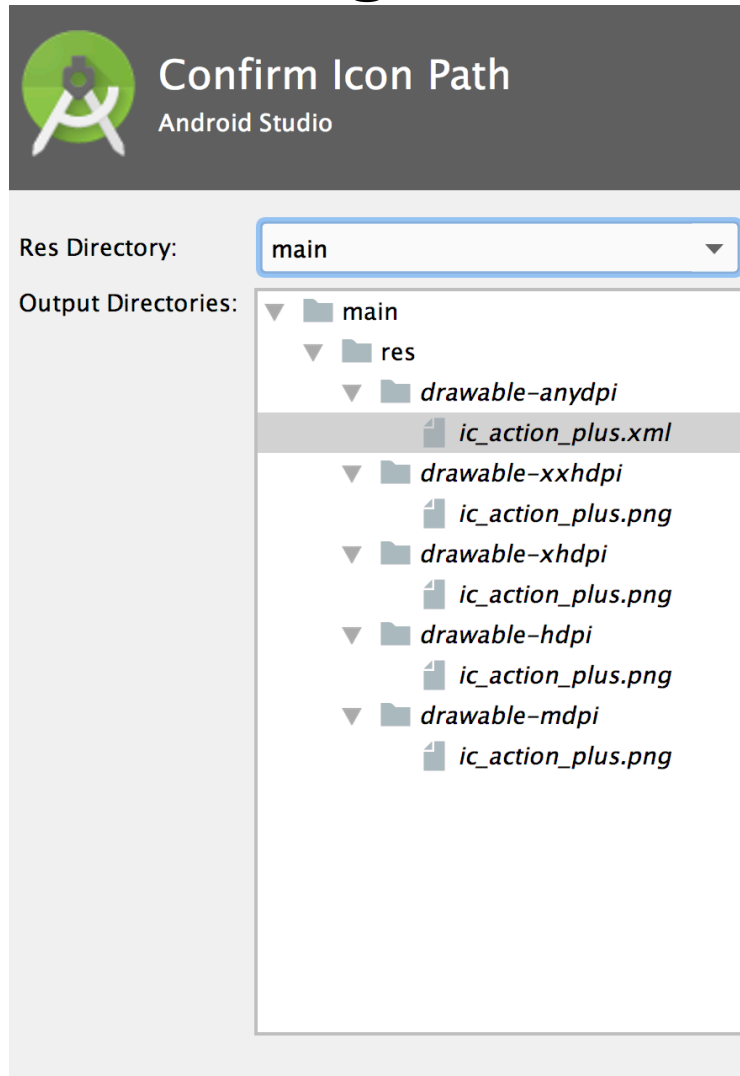
Padding: 0 %

Theme: **HOLO_DARK**

Preview: anydpi

Although the source image is white, if black is needed, just choose HOLO_LIGHT for theme

Adding icons to project



The included icon will be scaled and dropped into various **res/drawable** folders, one per screen density

Part 4:

Adding Icon to Action Bar/Launching AddEditMovie with ADD request code

Adding + icon to Action Bar

- This is a multi-step process:
 - Create a menu resource for the action bar, with the icon as a menu item
 - “Inflate” this menu resource in `Movies.java`, by overriding the callback method to draw the action bar that will be invoked when activity is launched
 - The menu will not be inflated in `AddEditMovie.java`, so that activity’s action bar will not have the add capability
 - In `Movies.java`, override the method that will be called when a menu item is clicked in the action bar, to handle the add event when + is clicked

Adding +: 1. Create a Menu Resource for Action Bar

- Create a folder called `menu` under `res`
- In the `res/menu` folder, create a menu resource file called `add_menu.xml`, with the following code:

```
<?xml version="1.0" encoding="utf-8"?>
<menu
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:appcompat="http://schemas.android.com/apk/res-auto"
  >
  <item android:id="@+id/action_add"
    android:icon="@drawable/ic_action_plus_white"
    android:title="@string/add"
    appcompat:showAsAction="always"/>
</menu>
```

↑
Meaning show at all
times, don't hide it
in overflow menu

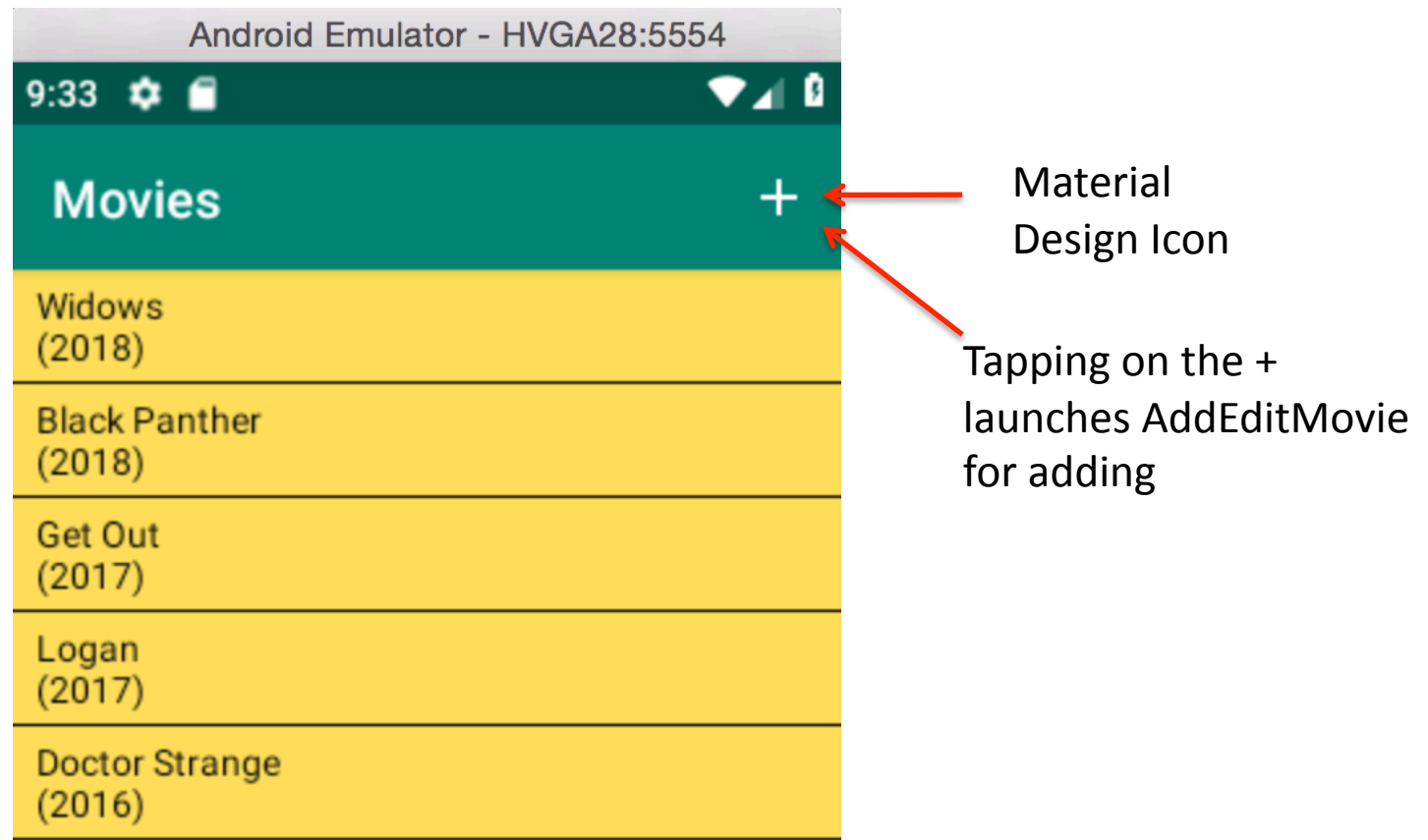
<https://developer.android.com/guide/topics/ui/menus.html>

Adding +: 2. Inflate Menu

- In `Movies.java`, override the `onCreateOptionsMenu(Menu)` method to inflate the menu resource – this method will be called when the app is launched

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.add_menu, menu);
    return super.onCreateOptionsMenu(menu);
}
```

+ Icon in Action Bar



Show/Edit and Add are done by the same activity, identified by different REQUEST codes sent by this parent activity

Adding +: 3. Override callback for event handling

- In `Movies.java`, override `onOptionsItemSelected` method (which is called whenever an item is clicked in the Action Bar):

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_add: ← Id of + icon in menu resource xml
            addMovie(); ← We will code this method to launch
                        AddEditMovie activity to add movie
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Launching Show Movie for Result, with request code for ADD

Movies.java

```
public static final int EDIT_MOVIE_CODE=1;
public static final int ADD_MOVIE_CODE=2;
...
public void addMovie() {
    Intent intent = new Intent(this, AddEditMovie.class);
    startActivityForResult(intent, ADD_MOVIE_CODE);
}
...
```

AddEditMovie finishes

- `onActivityResult` method distinguishes between return from Show/Edit and Add


```
@Override
protected void onActivityResult(int requestCode,
                                int resultCode,
                                Intent intent) {


    ...
    if (requestCode == EDIT_MOVIE_CODE) {
        Movie movie = movies.get(index);
        movie.name = name;
        movie.year = year;
        movie.director = director;

    } else if (requestCode == ADD_MOVIE_CODE) {
        movies.add(new Movie(name, year, director));
    }

    // redo the adapter to reflect change
    listView.setAdapter(
        new ArrayAdapter<Movie>(this,
                                R.layout.movie, movies));

    ...
}
```

 index of movie in movies list,
that was passed through to
AddEditMovie

 Adapter has to be redone
since source content has changed

Part 5:

Non-raw file I/O

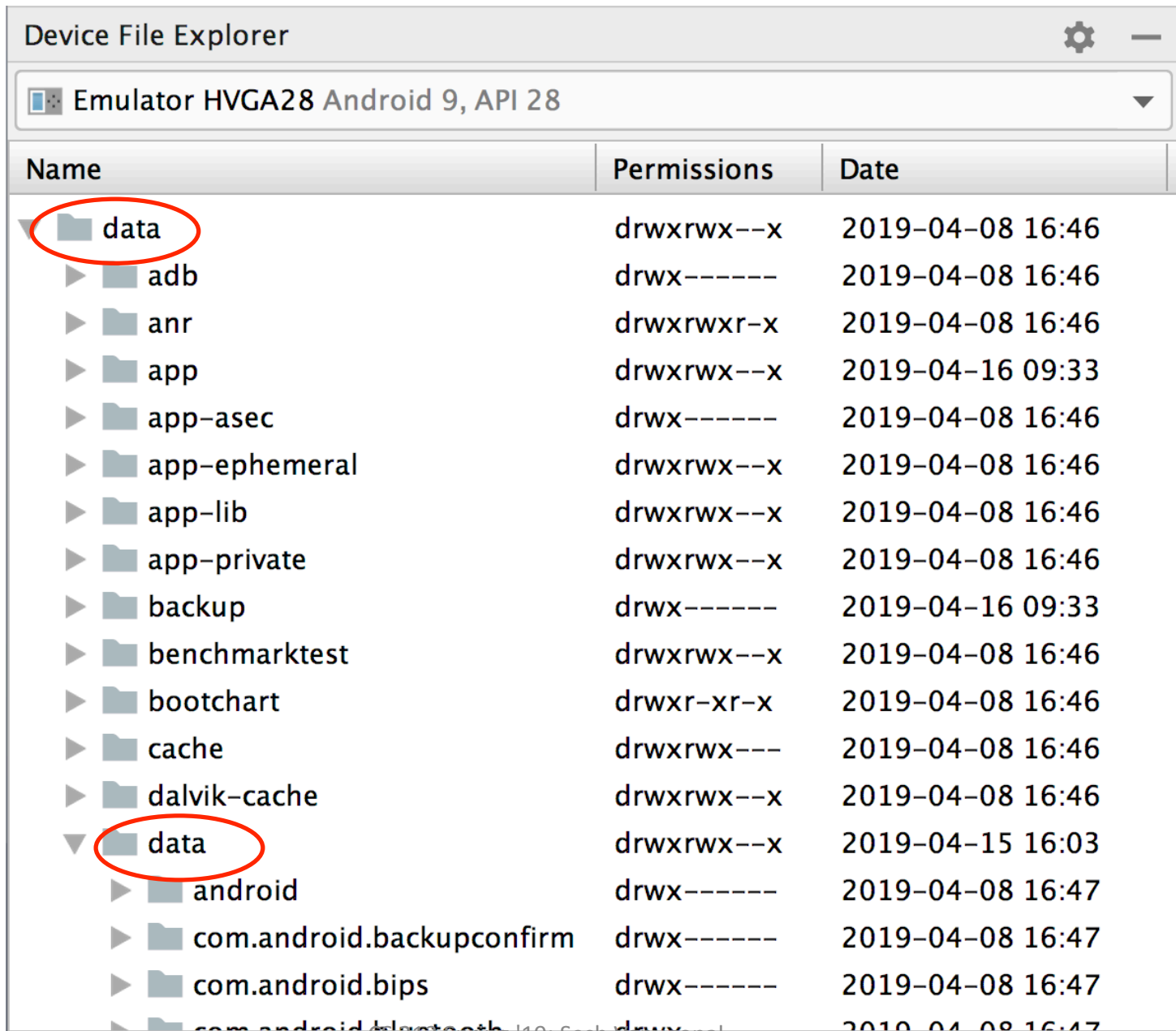
<https://developer.android.com/guide/topics/data/data-storage.html#filesInternal>

Movies Input List

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    ..
    // load movies from file, or if no file, from string array
    try {
        FileInputStream fis = openFileInput("movies.dat");
        BufferedReader br = new BufferedReader(
            new InputStreamReader(fis));
        String movieInfo=null; movies = new ArrayList<Movie>();
        while ((movieInfo = br.readLine()) != null) {
            String[] tokens = movieInfo.split("\\|");
            if (tokens.length == 3) {
                movies.add(new Movie(tokens[0],tokens[1], tokens[2]));
            } else { movies.add(new Movie(tokens[0], tokens[1])); }
        }
    } catch (IOException e) {
        // load from bootstrap list in string resources
        String[] moviesList = getResources().getStringArray(...);
        movies = new ArrayList<Movie>(moviesList.length);
        for (int i=0; i < moviesList.length; i++) {
            String[] tokens = moviesList[i].split("\\|");
            movies.add(new Movie(tokens[0],tokens[1]));
        }
    }
}
```


<https://developer.android.com/studio/debug/device-file-explorer>

Device File Explorer



Name	Permissions	Date
data	drwxrwx--x	2019-04-08 16:46
▶ adb	drwx-----	2019-04-08 16:46
▶ anr	drwxrwxr-x	2019-04-08 16:46
▶ app	drwxrwx--x	2019-04-16 09:33
▶ app-asec	drwx-----	2019-04-08 16:46
▶ app-ephemeral	drwxrwx--x	2019-04-08 16:46
▶ app-lib	drwxrwx--x	2019-04-08 16:46
▶ app-private	drwxrwx--x	2019-04-08 16:46
▶ backup	drwx-----	2019-04-16 09:33
▶ benchmarktest	drwxrwx--x	2019-04-08 16:46
▶ bootchart	drwxr-xr-x	2019-04-08 16:46
▶ cache	drwxrwx---	2019-04-08 16:46
▶ dalvik-cache	drwxrwx--x	2019-04-08 16:46
▼ data	drwxrwx--x	2019-04-15 16:03
▶ android	drwx-----	2019-04-08 16:47
▶ com.android.backupconfirm	drwx-----	2019-04-08 16:47
▶ com.android.bips	drwx-----	2019-04-08 16:47
▶ com.android.bluetooth	drwx-----	2019-04-08 16:47

Right click on files
for options to transfer
to and fro from local
filesystem

Device File Explorer		
Emulator HVGA28 Android 9, API 28		
Name	Permis...	Date
▶ com.android.wallpaper.livepicker	drwx----	2019-04-08 16:47
▶ com.android.wallpaperbackup	drwx----	2019-04-08 16:47
▶ com.breel.geswallpapers	drwx----	2019-04-08 16:47
▶ com.example.sesh.fc	drwx----	2019-04-08 16:47
▶ com.example.sesh.hello	drwx----	2019-04-09 17:35
▼ com.example.sesh.movies	drwx----	2019-04-16 10:42
▶ cache	drwxrws-	2019-04-15 16:03
▶ code_cache	drwxrws-	2019-04-15 16:03
▼ files	drwxrwx-	2019-04-15 16:03
? movies.dat	-rw-rw-r	2019-04-16 11:19
▶ com.example.sesh.runbbusroutes	drwx----	2019-04-11 17:28
▶ com.google.android.apps.docs	drwx----	2019-04-11 13:39
▶ com.google.android.apps.inputmetho	drwx----	2019-04-08 16:47
▶ com.google.android.apps.maps	drwx----	2019-04-08 16:49
▶ com.google.android.apps.messaging	drwx----	2019-04-08 16:47
▶ com.google.android.apps.nexuslaunc	drwx----	2019-04-08 16:47
▶ com.google.android.apps.photos	drwx----	2019-04-08 16:47
▶ com.google.android.apps.restore	drwx----	2019-04-08 16:47