

# CS 213 – Software Methodology

## Spring 2023

*Sesh Venugopal*

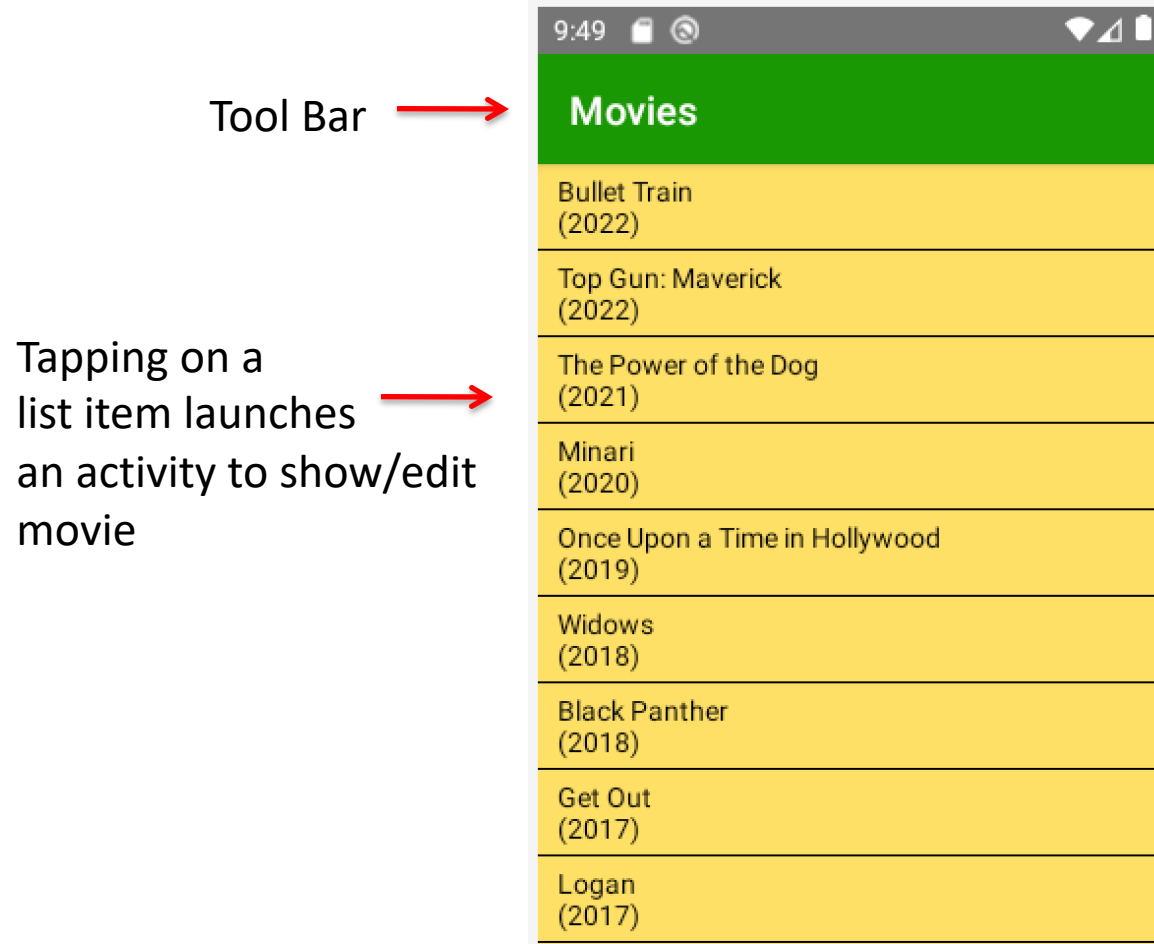
Apr 10

Android Programming

Activity for result | Dialogs | Menu & Icons | Device I/O

# Movies List Project

List set up like in the Rutgers Bus Routes app



# Part 1:

## Creating a New Activity for Result

# AddEditMovie Activity

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

Layout is in `add_edit_movie.xml`

A movie item was  
tapped in the parent  
movie list activity

The screenshot shows a mobile application interface for adding or editing a movie. At the top is a green header bar with a white back arrow and the text "Add/Edit Movie". Below the header, there are three text input fields. The first field is labeled "Name: (required)" and contains the text "The Power of the Dog". The second field is labeled "Year: (required)" and contains the text "2021". The third field is labeled "Director:" and is currently empty. At the bottom of the form, there are two yellow buttons: "SAVE" and "CANCEL".

In layout xml event handler:  
`android:onClick="save"`

Method name

In layout xml event handler:  
`android:onClick="cancel"`

Method name

# Implementing activity AddEditMovie

File -> New -> Activity -> Empty Activity

## Empty Activity

Creates a new empty activity

Activity Name

AddEditMovie

☐

Generate a Layout File

☐

Launcher Activity

Don't generate a layout file. Instead, in the generated AddEditMovie class, setContentView to R.layout.add\_edit\_movie.xml

Package name

com.example.movies



Source Language

Java



# AddEditMovie Class

```
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 (when called for edit)

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

```
    ...
```

```
}
```

↑  
Text fields in fill out form

in `onCreate` →

```
    Toolbar myToolbar = (Toolbar) findViewById(R.id.my_toolbar);  
    myToolbar.setTitle("Add/Edit Movie");  
    setSupportActionBar(myToolbar);  
    getSupportActionBar().setDisplayHomeAsUpEnabled(true);
```

in Manifest → `android:parentActivityName="com.example.movies.Movies"`

# AddEditMovie Class

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

```
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

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();  
    String year = movieYear.getText().toString();  
    String director = movieDirector.getText().toString();  
  
    // 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 info back to caller (activity that launched this activity)  
    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

# Part 2:

## Register Activity for Result

# Register Activity for Result

<https://developer.android.com/training/basics/intents/result> (see bottom of page)

The Add/Edit activity will return a result when Save button is clicked, and the result needs to be communicated back to the launching activity (Movies).

This handshake needs to be registered

## Movies.java

```
protected void onCreate(Bundle savedInstanceState) {  
    ...  
    listView.setOnItemClickListener((list, view, pos, id) -> showMovie(pos));  
  
    // register add/edit activities in onCreate  
    // registration must be done before creation is completed  
    registerActivities();  
}  
  
public void registerActivities() {  
    ActivityResultLauncher<Intent> startForResultEdit =  
        registerForActivityResult(new ActivityResultContracts.StartActivityForResult(),  
        new ActivityResultCallback<ActivityResult>() {  
            @Override  
            public void onActivityResult(ActivityResult result) {  
                if (result.getResultCode() == Activity.RESULT_OK) {  
                    // handle result  
                    ...  
                }  
            }  
        });  
}
```

 This code will be set in ShowMovie

# Register Activity for Result

`ActivityResultCallback` is a functional interface, so we can use lambda – each of the edit and add functions get's its own registration instance

`Movies.java`


---

```
...
private ActivityResultLauncher<Intent> startForResultEdit;
private ActivityResultLauncher<Intent> startForResultAdd;

public void registerActivities() {

    startForResultEdit =
        registerForActivityResult(new ActivityResultContracts.StartActivityForResult(),
            result -> {
                if (result.getResultCode() == Activity.RESULT_OK) {
                    applyEdit(result, "edit");
                }
            });

    startForResultAdd =
        registerForActivityResult(new ActivityResultContracts.StartActivityForResult(),
            result -> {
                if (result.getResultCode() == Activity.RESULT_OK) {
                    applyEdit(result, "add");
                }
            });
}
```

  
back from edit

  
back from add

---

# Part 3:

## Process the result

## applyEdit method

Movies.java

---

```
...
private void applyEdit(ActivityResult result, String addEdit) {
    Intent intent = result.getData();
    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);

    if (addEdit.equals("edit")) {
        // update the movie
        Movie movie = movies.get(index);
        movie.name = name;
        movie.year = year;
        movie.director = director;
    } else if (addEdit.equals("add")){
        movies.add(new Movie(name, year, director));
    }

    // redo the adapter to reflect change ← Unlike FX ListView/ObservableArrayList
    listView.setAdapter(
        new ArrayAdapter<Movie>(this, R.layout.movie, movies));
}
```

---

# Part 4:

## Launch activity for result

# Launch AddEditMovie for edit or add

Movies.java

---

...

```
private 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);
```

```
    // launch for edit
```

```
    Intent intent = new Intent(this, AddEditMovie.class);
```

```
    intent.putExtras(bundle);
```

```
    startForResultEdit.launch(intent);
```

```
}
```

```
private void addMovie() {
```

*This method will be called when a movie is added, which will happen when a '+' icon is clicked – coming up a little later*

```
    // launch for add
```

```
    Intent intent = new Intent(this, AddEditMovie.class);
```

```
    startForResultAdd.launch(intent);
```

```
}
```

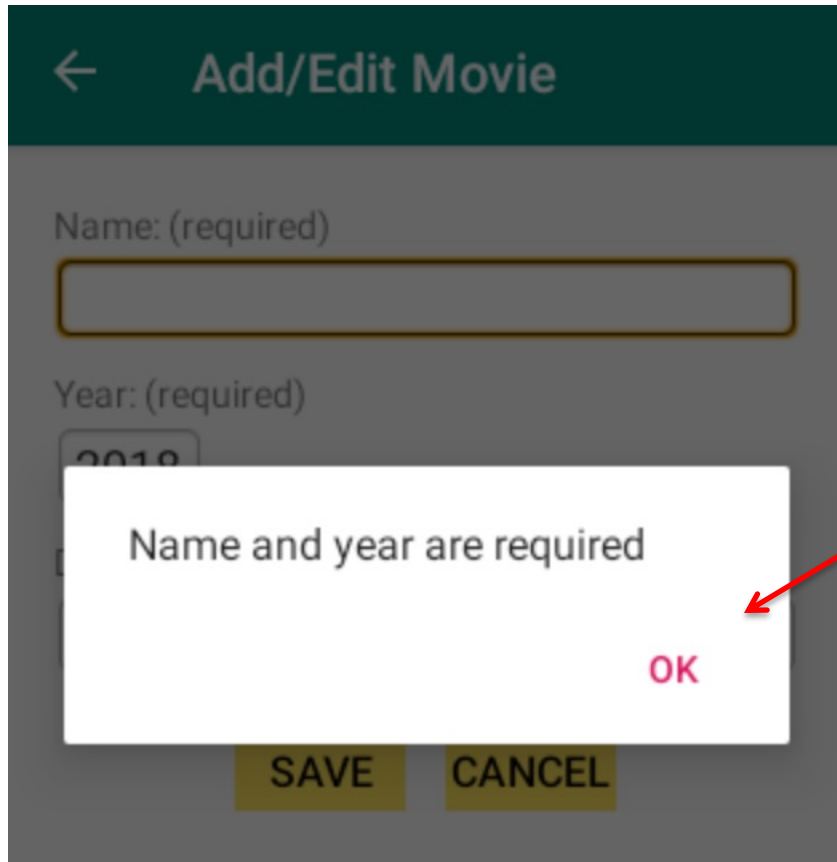
**TRY OUT THE APP (for edit) !**



# Part 5: 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>

```
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:

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

*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*

# 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) {  
    // gather all data from text fields  
    ...  
  
    // pop up dialog if errors in input, and return  
    // 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.show(getSupportFragmentManager(), "badfields");  
        return; // does not quit activity, just returns from method  
    }  
    ...  
    // make Bundle  
    ...  
}
```

**TRY OUT THE APP (for dialog on bad edit) !**

# Part 6:

## 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 Toolbar
- There are prefab icons supplied by the Android guys for a whole lot of standard tasks, including one to add content (such as movies in our app)

Go to [Google fonts \(https://fonts.google.com/icons\)](https://fonts.google.com/icons)

Under the “UI actions” section, click on the '+' icon - this slides out a drawer.

Select the Web tab, then download PNG. This will download a file named <whatever>.png

# Adding icons to project

Right click on **res**, then choose **New -> Image Asset**, then configure like this:

Name that will be used in app

The downloaded png

Source image is black, but we need white, so choose HOLO\_DARK theme

Configure Image Asset

Icon Type: Action Bar and Tab Icons

Name: ic\_action\_plus

Asset Type: ☒ Image ☐ Clip Art ☐ Text

Path: ld\_FILL0\_wght400\_GRAD0\_opsz48.png

Trim: ☐ Yes ☒ No

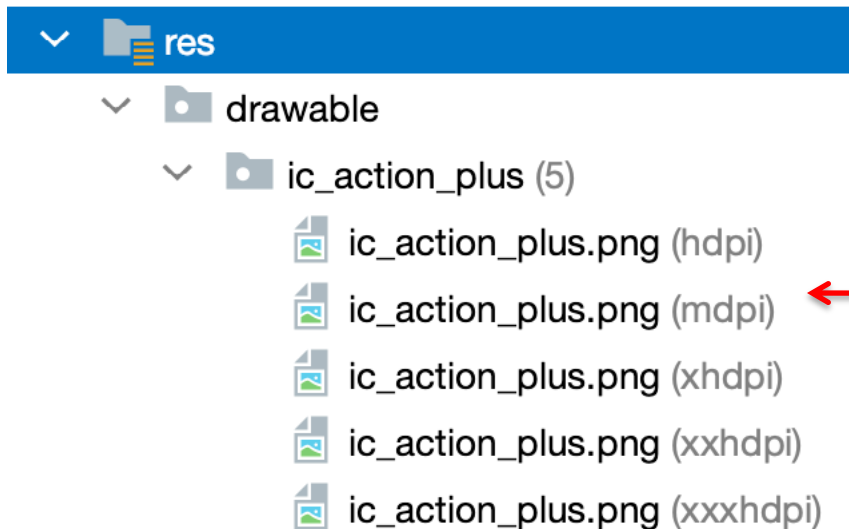
Padding: 0 %

Theme: HOLO\_DARK

Preview: xxxhdpi



# Adding icons to project



The included icon will be scaled to sizes required various device resolutions and dropped into `res/drawable`

# Part 7:

## Adding Icon to Toolbar

# Adding + icon to Toolbar

- 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 that draws the action bar 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

# 1. Create a Menu Resource for Action Bar

- Create a folder (directory) 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:app="http://schemas.android.com/apk/res-auto">
  <item android:id="@+id/action_add"
        android:icon="@drawable/ic_action_plus"
        android:title="@string/menu_add"
        app:showAsAction="always" />
</menu>
```



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

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

## 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) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.add_menu, menu);
    return true;
}
```

# + Icon in Toolbar



Clicking on the +  
launches [AddEditMovie](#)  
for 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(); ← This method was coded earlier
                           (Slide 16)

            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

TRY OUT THE APP (for adding a movie) !

# Part 8:

## Non-raw file I/O

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




# Sample `movies.dat` file

A Quiet Place Part II|2021  
American Sniper|2014  
Black Panther: Wakanda Forever|2022|Ryan Coogler  
Bohemian Rhapsody|2018  
Conan the Barbarian|2011  
Doctor Strange|2016  
Get Out|2017  
Interstellar|2014  
Jason Bourne|2016  
Logan|2017|James Mangold  
Mad Max:Fury Road|2015  
Nightcrawler|2014  
Nomadland|2020|Chloe Zhao  
Once Upon a Time in Hollywood|2019  
Selma|2014  
Sicario|2015  
Sound of Metal|2020  
Straight Outta Compton|2015  
Sully|2016  
The Equalizer|2014  
The Gift|2010  
The Martian|2015  
Top Gun: Maverick|2022|Joseph Kosinski  
Training Day|2001

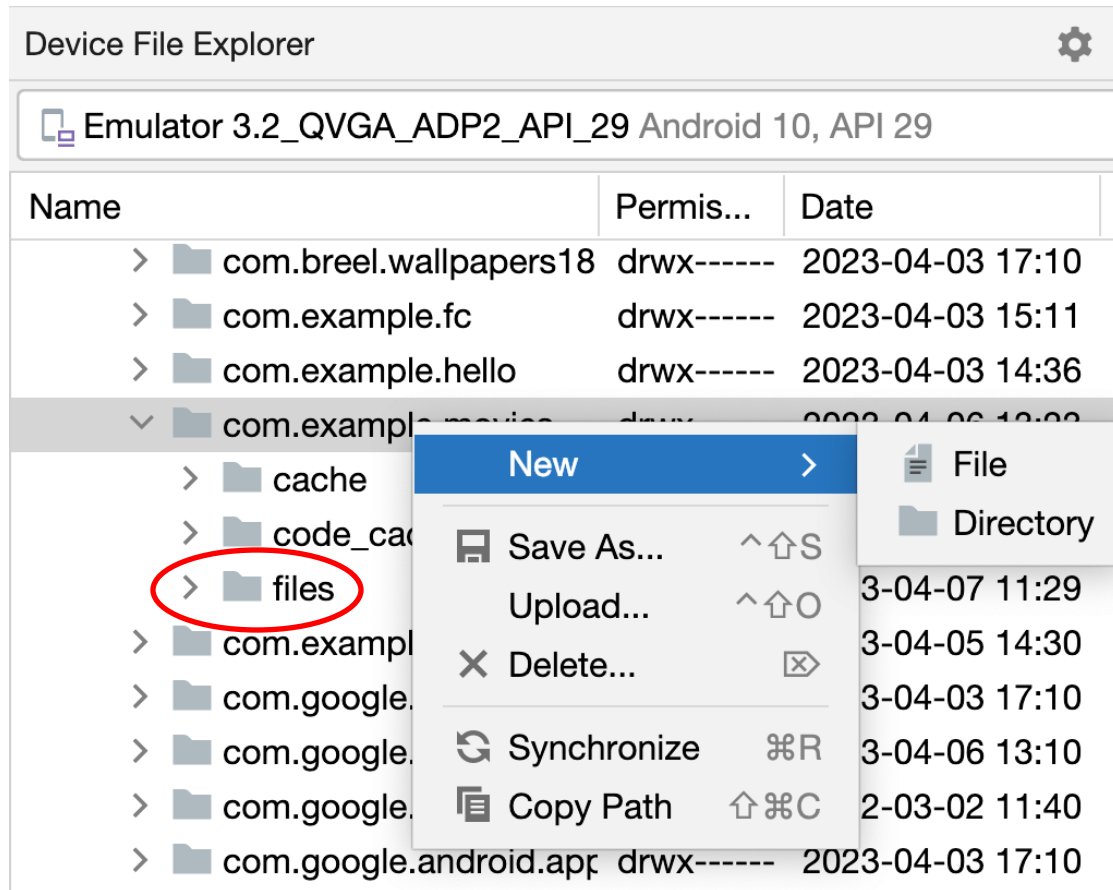
# Movies Input List

```
@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 stock 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]));
        }
    }
    ...
}
```

# Device File Explorer

Device File Explorer 		
Emulator 3.2_QVGA_ADP2_API_29 Android 10, API 29		
Name	Permis...	Date
> acct	dr-xr-xr-x	2023-04-03 17:10
> apex	drwxr-xr-x	2023-04-03 17:10
> bin	lrw-r--r--	2021-08-02 12:20
> cache	drwxrwx---	2021-08-02 11:40
> config	drwxr-xr-x	2023-04-03 17:10
> d	lrw-r--r--	2021-08-02 12:20
✓ data	drwxrwx--;	2022-03-02 11:40
> adb	drwx-----	2022-03-02 11:40
> anr	drwxrwxr->	2022-03-02 11:40
> apex	drwxr-x---	2022-03-02 11:40
> app	drwxrwx--;	2023-04-07 11:18
> app-asec	drwx-----	2022-03-02 11:40
> app-ephemeral	drwxrwx--;	2022-03-02 11:40
> app-lib	drwxrwx--;	2022-03-02 11:40
> app-private	drwxrwx--;	2022-03-02 11:40
> app-staging	drwxr-x---	2022-03-02 11:40
> backup	drwx-----	2023-04-07 11:18
> bootchart	drwxr-xr-x	2022-03-02 11:40
> cache	drwxrwx---	2022-03-02 11:40
> dalvik-cache	drwxrwx--;	2022-03-02 11:40
✓ data	drwxrwx--;	2023-04-06 13:23
> android	drwx-----	2023-04-03 17:10
> com.android.apps.tag	drwx-----	2022-03-02 11:40

Right click on `com.example.movies`, and make a new directory named `files` under it



Device File Explorer		
Emulator 3.2_QVGA_ADP2_API_29 Android 10, API 29		
Name	Permis...	Date
> com.breel.wallpapers18	drwx-----	2023-04-03 17:10
> com.example.fc	drwx-----	2023-04-03 15:11
> com.example.hello	drwx-----	2023-04-03 14:36
✓ com.example.movies	drwx-----	2023-04-06 13:23
> cache	drwxrws--	2023-04-06 13:23
> code_cache	drwxrws--	2023-04-06 13:23
> files		2023-04-07 11:29
> com.ex		2023-04-05 14:30
> com.g		2023-04-03 17:10
> com.g		2023-04-06 13:10
> com.g		2022-03-02 11:40
> com.g		2023-04-03 17:10
> com.g		2023-04-03 17:10
> com.g		2023-04-03 17:10
> com.g		2023-04-03 17:10
> com.google.android.app	drwx-----	2022-03-02 14:07

Right click on files  
for options for transfer  
to (Save As ...)  
or from (Upload...)  
local filesystem

Uploaded  
movies.dat

✓ files	drwxrwxrw	2023-04-07
movies.dat	-rwxrwxrw	2023-04-07

Run the app – now the movies are read in from `movies.dat` file

