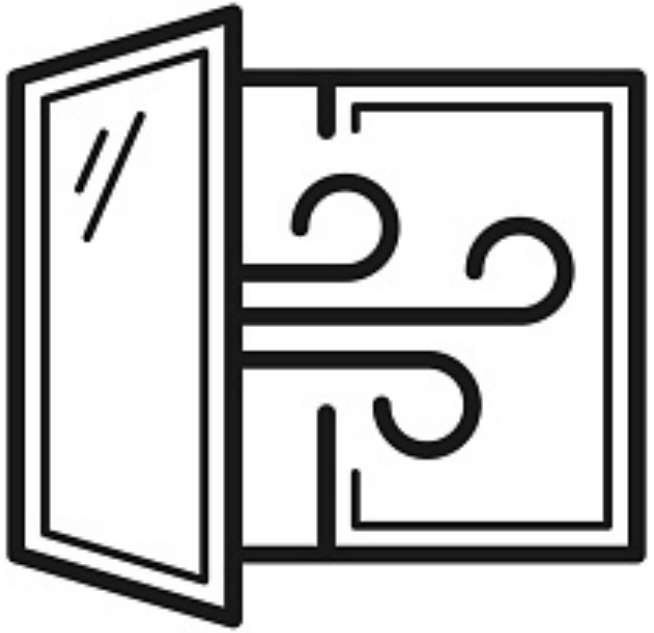
Android App Programming

Lecture 7: UX Programming
Nov 30th, 2021

Anastasia Bezerianos
Université Paris-Saclay

in class



Last lecture recap

SQLite

- DBMS integrated into Android: SQLite
- packaged into the app
  - runs in the same process
  - no connection setup necessary

- use the class `SQLiteDatabase` included in Android

UX Programming

UX Design

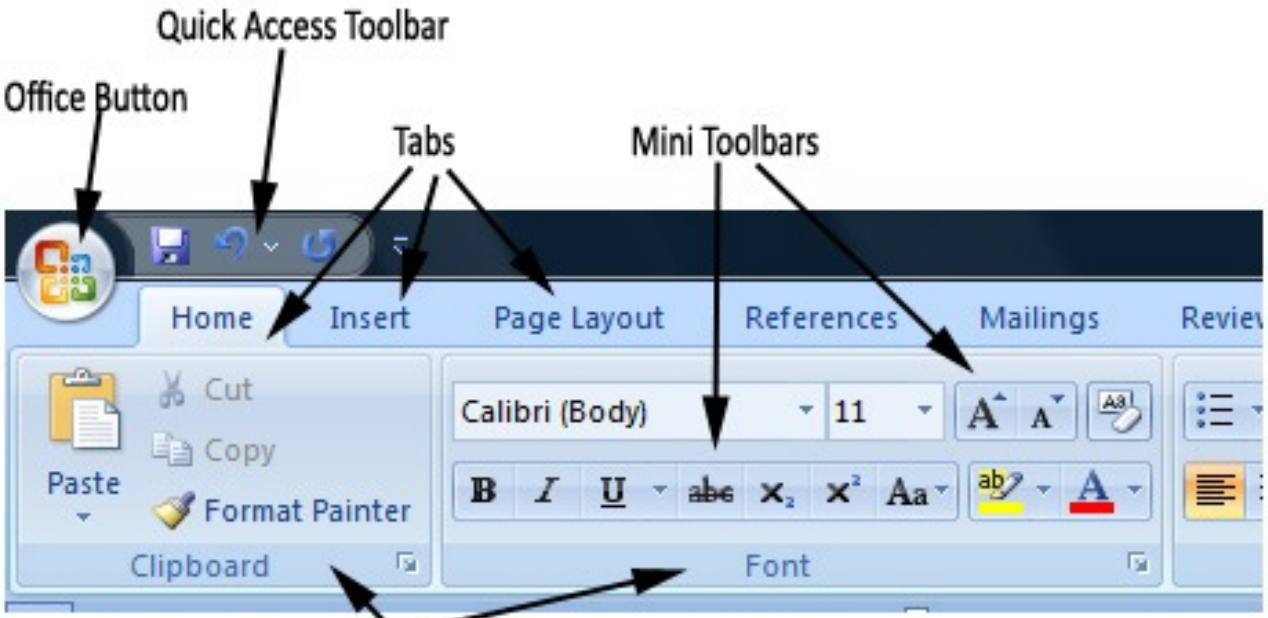
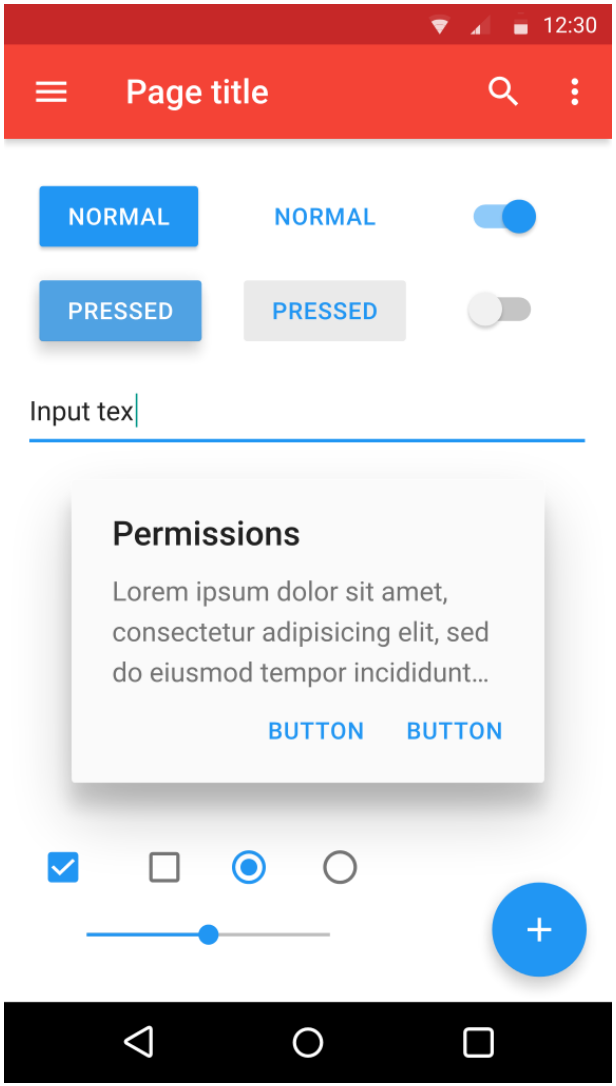


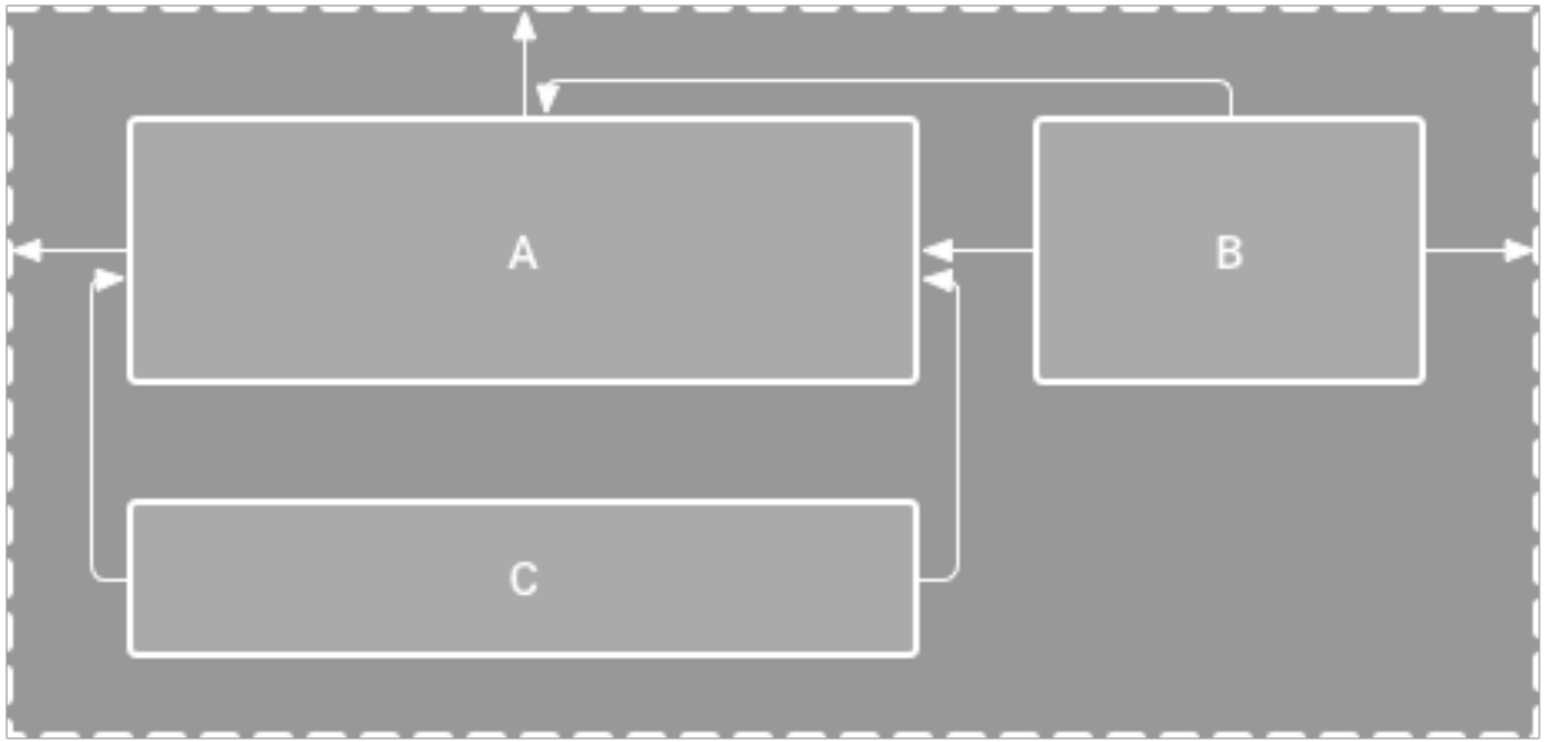
Figure 1: The Ribbon Interface

Material Design

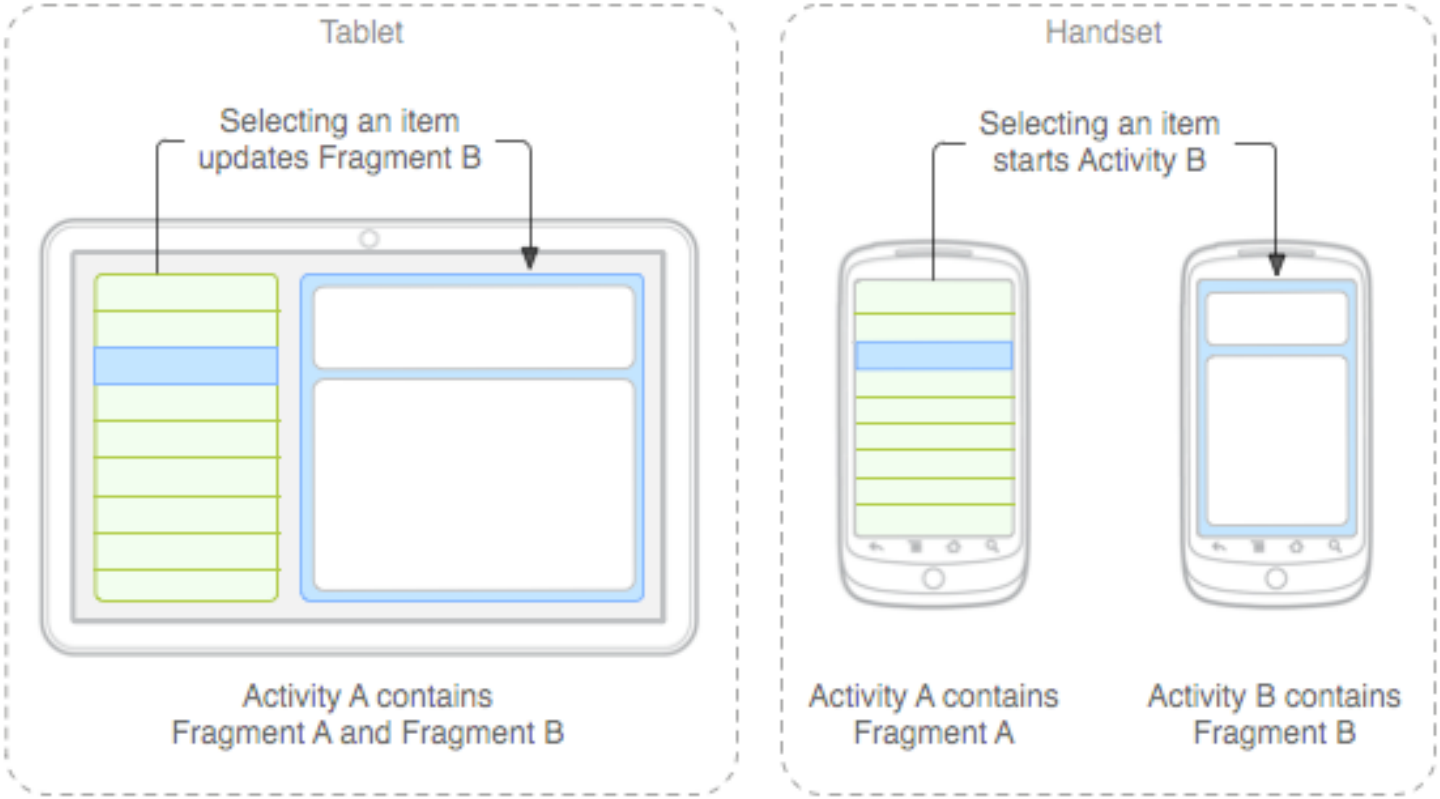


https://material.io/design

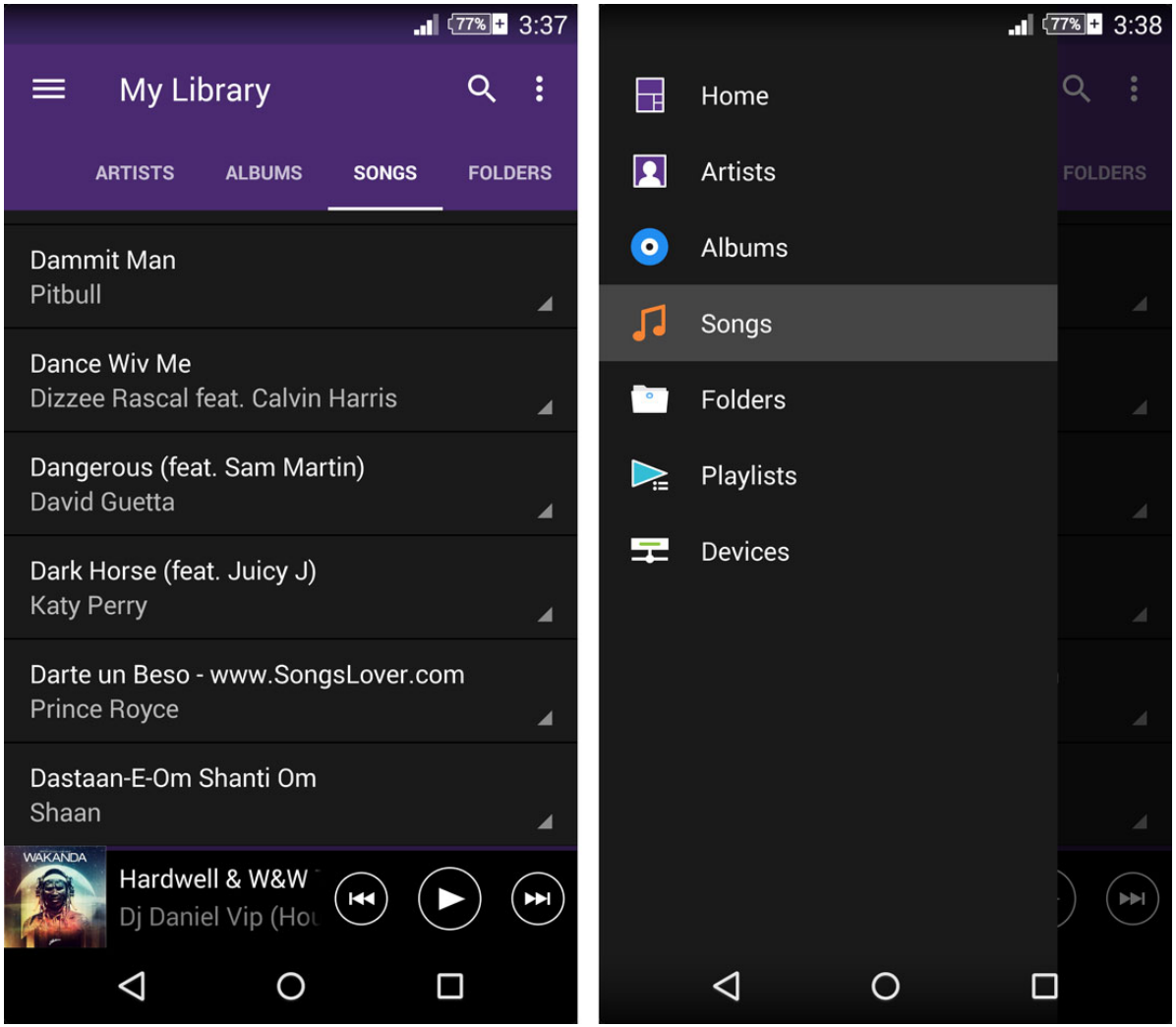
Constraint Layout



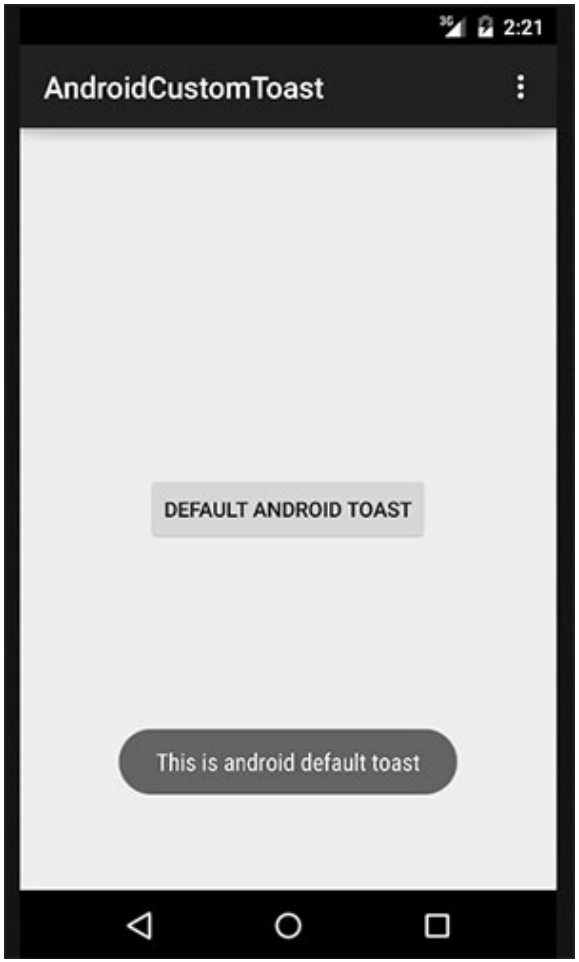
Fragments

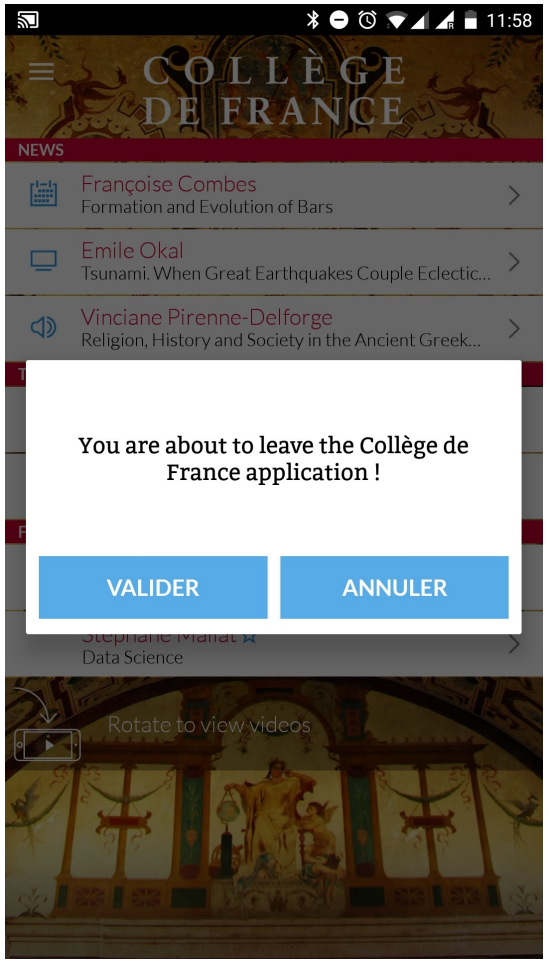

Tablet
Selecting an item
updates Fragment B
Activity A contains
Fragment A and Fragment B
Handset
Selecting an item
starts Activity B
Activity A contains
Fragment A
Activity B contains
Fragment B

Hamburger Menus



Notifications and Dialogs





Layouts

- XML layouts decouple View (visual components) and data treatment (which is good! More next week)
- but cannot be used to create dynamic user interfaces (e.g., interface changes based on external factors, lists of choices that change size, …)

Layouts

- We organise Views (eg radio buttons, buttons, text fields, image views) using:
  - A layout: general positioning of Views
  - A constraint set: specific relationships between Views

Layouts, Constraint Sets

- We organise Views (eg radio buttons, buttons, text fields, image views) using:
  - A layout: general positioning of Views
  - A constraint set: specific relationships between Views

Common Layouts

- Constraint Layout
- LinearLayout arranges the items in a one-dimensional list
- Table or GridLayout arranges all items in a grid
- FrameLayout usually to frame a single item

Lab 7

- Topic: Dynamic Layouts and Event Programming

Lab 7

- Ex1 part 1 & 2: create a simple interface programmatically

  Follow the example in Chapter 23.

  Run the program with each addition to see how your interfaces changes.

  Then change the EditText to a TextView

  (Stop video to do the example)

Ex1 part 1 & 2 solution

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        configureLayout();
    }

    private int convertToPx(int value) {
        Resources r = getResources();
        int px = (int) TypedValue.applyDimension(
                TypedValue.COMPLEX_UNIT_DIP, value,
                r.getDisplayMetrics());
        return px;
    }

    private void configureLayout() {
        Button myButton = new Button(this);
        myButton.setText(getString(R.string.press_me));
        myButton.setBackgroundColor(Color.YELLOW);
        myButton.setId(R.id.myButton);

        TextView myEditText = new TextView(this);
        myEditText.setId(R.id.myEditText);
        myEditText.setText(R.string.default_text);
        int px = convertToPx(200);
        myEditText.setWidth(px);

        ConstraintLayout myLayout = new ConstraintLayout(this);
        myLayout.setBackgroundColor(Color.rgb(3, 132, 252));
        myLayout.addView(myButton);
        myLayout.addView(myEditText);

        setContentView(myLayout);

        ConstraintSet set = new ConstraintSet();

        set.constrainHeight(myButton.getId(),
                ConstraintSet.WRAP_CONTENT);
        set.constrainWidth(myButton.getId(),
                ConstraintSet.WRAP_CONTENT);
        set.connect(myButton.getId(), ConstraintSet.LEFT,
                ConstraintSet.PARENT_ID, ConstraintSet.LEFT, 0);
        set.connect(myButton.getId(), ConstraintSet.RIGHT,
                ConstraintSet.PARENT_ID, ConstraintSet.RIGHT, 0);
        set.connect(myButton.getId(), ConstraintSet.TOP,
                ConstraintSet.PARENT_ID, ConstraintSet.TOP, 0);
        set.connect(myButton.getId(), ConstraintSet.BOTTOM,
                ConstraintSet.PARENT_ID, ConstraintSet.BOTTOM, 0);

        set.constrainHeight(myEditText.getId(),
                ConstraintSet.WRAP_CONTENT);
        set.constrainWidth(myEditText.getId(),
                ConstraintSet.WRAP_CONTENT);
        set.connect(myEditText.getId(), ConstraintSet.LEFT,
                ConstraintSet.PARENT_ID, ConstraintSet.LEFT, 0);
        set.connect(myEditText.getId(), ConstraintSet.RIGHT,
                ConstraintSet.PARENT_ID, ConstraintSet.RIGHT, 0);
        set.connect(myEditText.getId(), ConstraintSet.BOTTOM,
                myButton.getId(), ConstraintSet.TOP, 70);


        set.applyTo(myLayout);
    }
}
```

Helper method to convert density independent pixels (automatically scaled to match the device display at application runtime) to actual screen pixels

Helper method to draw Views

Create a button, give it a text and set its color.
The setId connects it to an ID that is in the xml resources (so that it is unique and accessible for layouts). You need to add it to res > values > id with <item name="myButton" type="id" />

Create a TextView, give it a default text and size. Again setId connects it to the xml resources. You need to add it to res > values > id with <item name="myEditText" type="id" />

Create the myLayout (a ConstraintLayout) and add the button and text view to it

Tell the Activity to show the layout myLayout

Create a ConstraintSet

apply constraints for the button: constraint all sizes of the button to all sizes of the parent window (ie put it in the middle of window).

apply constraints for the text view: as with button for most sides, but the Bottom part of it is constraint by the Top of the button (ie it is on top of the button) with a buffer of 70 pixels

apply the ConstraintSet to the layout

Event Programming

- Paradigm: flow of program determined by events
- Events are user actions (e.g., mouse clicks, keyboard presses), or msg from programs or threads (e.g., reading data from the web)
- Code for handling actions in Event Listeners (e.g., onClick), Callback methods, or Observable objects

Common Listeners

onClickListener – When click detected on a View. Corresponds to the *onClick()* callback method

onLongClickListener – When the user maintains the touch over a view for an extended period. Corresponds to the *onLongClick()* callback.

onTouchListener – Used to detect any form of contact with the touch screen including individual or multiple touches and gesture motions (see Ch 26, 27).

onFocusChangeListener – Detects when focus moves away from the current view as the result of interaction with a track-ball or navigation key. Corresponds to the *onFocusChange()* callback method.

onKeyListener – Used to detect when a key on a device is pressed while a view has focus. Corresponds to the *onKey()* callback method.

OnItemClickListener - Callback method to be invoked when an item in a list or collection has been clicked.

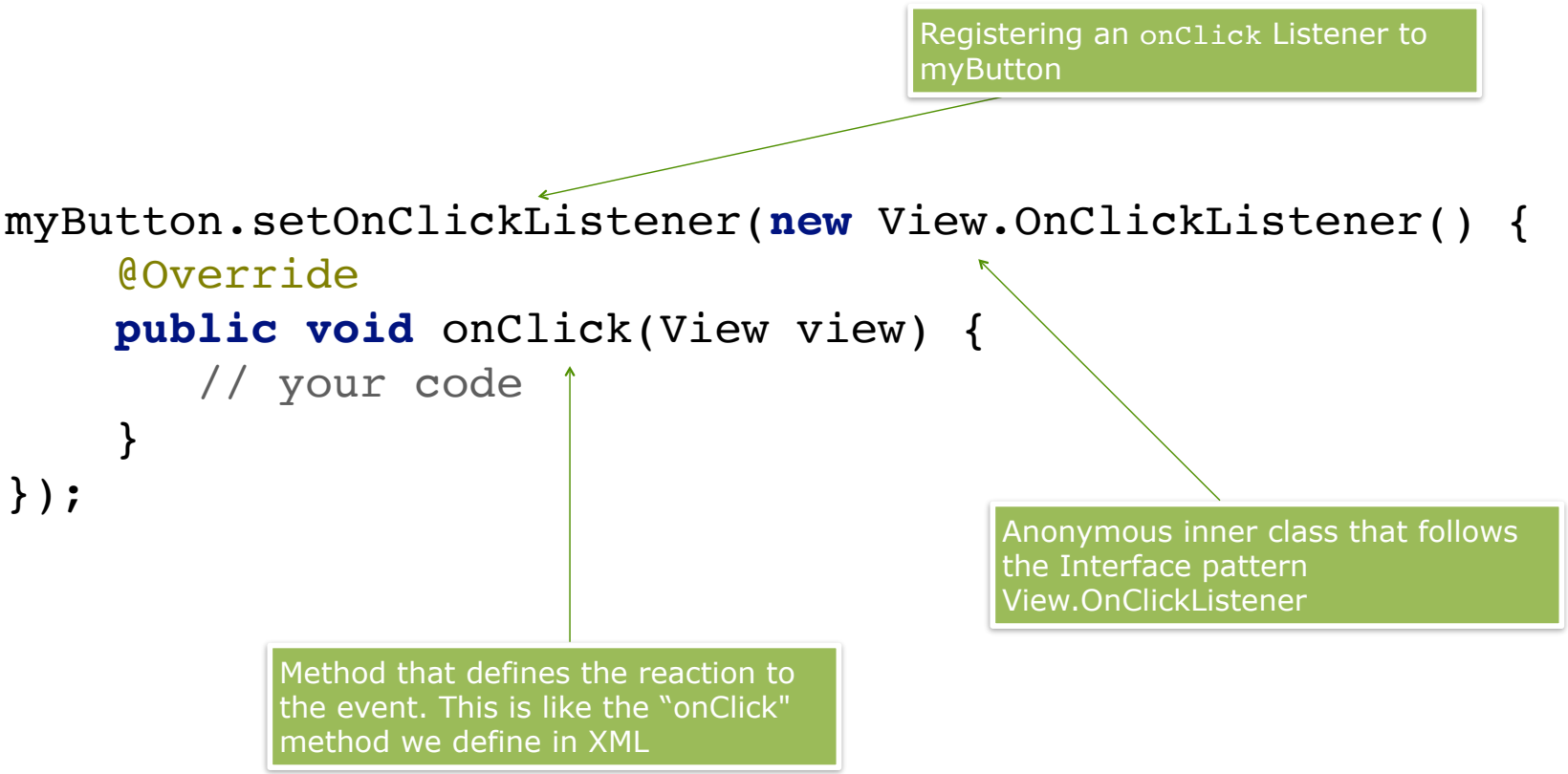
Adding a listener

- So far we have “registered" Listeners in XML through the Properties or in code (onClick)
- And implemented their code in our activity class
- Next, we will see how to add a listener to a View defined programmatically (as in our previous examples)

Adding a listener

- To handle an event, a View must
  - register an Event Listener
    - using methods setEventListener (eg. `setOnClickListener`, `setOnItemClickListener` ...)
  - implement the required Listener methods
    - objects of a class that extends (inherits) from a Java interface pattern. This class will have an onEvent method you need to provide code for (eg. `onClick`)

Adding a listener



Lab 7

- Ex1 part 3: add a listener

Following Ch 25 (in particular 25.6)

Ex1 part 3

```
public class MainActivity extends AppCompatActivity {

    …
    …


    private void configureLayout() {

        …

        Button myButton = new Button(this);
        …

        EditText myEditText = new EditText(this);
        …


        myButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                myEditText.setText("Button Clicked");
            }
        });

        myButton.setOnLongClickListener(new View.OnLongClickListener() {
            @Override
            public boolean onLongClick(View view) {
                myEditText.setText("Long Button Clicked");
                return false;
            }
        });


        ConstraintLayout myLayout = new ConstraintLayout(this);
        myLayout.setBackgroundColor(Color.rgb(3, 132, 252));
        myLayout.addView(myButton);
        myLayout.addView(myEditText);

        …

    }
}
```

Add an OnClickListener to button

Anonymous inner class that follows the Interface pattern View.OnClickListener

onClick method with our code

Anonymous inner class that follows the Interface pattern View.OnLongClickListener

onLongClick method with our code

Layouts

- You can use other layouts and add Views programmatically
- eg Linear Layout

```
LinearLayout group = new LinearLayout(this);
group.setOrientation(VERTICAL);
```

- Grid Layout

```
GridLayout group = new GridLayout(this);
group.setColumnCount(3);
```

- Then add Views programmatically

```
group.addView(new Button(this));
```

Layouts

- Add the layouts to your main view, for example `myLayout.addView(group);`

- You can also change their the layout position constraints in the same way as other Views

```
set.constrainHeight(group.getId(),
        ConstraintSet.WRAP_CONTENT);
set.constrainWidth(group.getId(),
        ConstraintSet.WRAP_CONTENT);
set.connect(group.getId(), ConstraintSet.RIGHT,
        ConstraintSet.PARENT_ID, ConstraintSet.RIGHT, 50);
set.connect(group.getId(), ConstraintSet.LEFT,
        ConstraintSet.PARENT_ID, ConstraintSet.LEFT, 50);
```

Lab 7

- Ex2
  - Part 1: experiment with the Constraint layout
  - Part 2: change the EditText to TextView in the code
- Part 3 & 4: create a Grid Layout and add buttons to it (see hint in next page)

Ex2: A Grid Layout

```
…. // previous part of Ex 1

GridLayout group = new GridLayout(this);
group.setColumnCount(3);

for (int i=1;i<10;++i) {
    Button b = new Button(this);
    b.setText("hello");
    group.addView(b);

    b.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View view) {
            myEditText.setText("other buttons Clicked");
        }
    });
}

group.setId(R.id.group);

myLayout.addView(group);
```

Creating a Grid Layout with 3 columns

Add 10 buttons to it

Add an OnClickListener to each one of these buttons

Make sure you've added this in res > values > id.xml

adding it to the main ConstraintLayout

Ex2: A Grid Layout with images

```
…. // previous part of Ex 1

GridLayout group = new GridLayout(this);
group.setColumnCount(3);

for (int i=1;i<10;++i) {
    ImageView imageView = new ImageView(this);
    imageView.setImageResource(R.drawable.sample);
    group.addView(imageView);

    imageView.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View view) {
            myEditText.setText("Image Clicked");
        }
    });
}

group.setId(R.id.group);

myLayout.addView(group);
```

Add 10 Image Views

This image "sample.jpg" should exist under res > drawable

Add an OnClickListener to each one of these ImageViews

Dynamic Layouts

- What happens when we have too many buttons to add ?

- Two options:
  - Create a ScrollView and add the Layout inside it
  - Use an Adapter View

ScrollView

- Can be added inside a ConstraintLayout
- Can add a layout inside it (to make it scrollable)
- For example:

```
ScrollView scroll_group = new ScrollView(this);
scroll_group.addView(group);
```

- Then add the ScrollView to the main layout

Lab 7

- Ex2 part 4

```
public class MainActivity extends AppCompatActivity {

    …….
  // previous part of Ex 2

    private void configureLayout() {

        });

        ConstraintLayout myLayout = new ConstraintLayout(this);
     …


        GridLayout group_layout = new GridLayout(this);
     …

        ScrollView scroll_group = new ScrollView(this);
        scroll_group.addView(group_layout);


        myLayout.addView(scroll_group);

        set.applyTo(myLayout);
    }


}
```

ScrollView

I add in it my grid layout

Now I add the scroll view to my main layout (instead of my grid layout)

Adapter View

- Views that adapt to the number of items they contain.
- They read the Views to display from a list or DB.
- Examples include
  `ListView`, `GridView`, `Spinner`, `Gallery`
- Adapter Views need an Adapter class (that extends **`BaseAdapter`**) that defines how to read the items to display

Lab 7

- Ex3
- following this tutorial https://www.androidhive.info/2012/02/android-gridview-layout-tutorial/

ImageAdapter

```
public class ImageAdapter extends BaseAdapter {

    private Context mContext;

    // Keep all Images in array
    public Integer[] mThumbIds = {
            R.drawable.boudin, R.drawable.degas, R.drawable.francis, R.drawable.gorky,
            R.drawable.hassam, R.drawable.krasner, R.drawable.monet, R.drawable.pissaro,
            R.drawable.poons, R.drawable.renoir, R.drawable.rothko, R.drawable.sisley
    };


  // Constructor
    public ImageAdapter(Context c){
      mContext = c;
    }

    @Override
    public int getCount() {
        return mThumbIds.length;
    }

    @Override
    public Object getItem(int pos_i) {
        return mThumbIds[pos_i];
    }

    @Override
    public long getItemId(int i) {
        return 0;
    }

    @Override
    public View getView(int i, View view, ViewGroup viewGroup) {
        ImageView imageView = new ImageView(mContext);
        imageView.setImageResource(mThumbIds[i]);
        imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
        imageView.setLayoutParams(new GridView.LayoutParams(70, 70));
        return imageView;
    }
}
```

Adapter class

Keeping the context so I know where to draw the Views on

List of images to draw, they are stored under res > drawable

Constructor that stores the context to draw on

Length of objects handled by the Adapter (in our case our list of image sources)

Get item at position pos_i

We are not storing IDs so this does not do anything really

Tells the GridView what to draw in position ***i*** of the grid (in our case an ImageView that loads the image at i)
...
and how to draw it (we scale it to 70,70 pixels)

main_activity.xml

My layout is slightly different from that of the tutorial:
I use a ConstraintLayout and inside it I add a GridView.
Here is the xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://
schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <GridView xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/grid_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:columnWidth="100dp"
        android:numColumns="auto_fit"
        android:verticalSpacing="10dp"
        android:horizontalSpacing="10dp"
        android:stretchMode="columnWidth"
        android:gravity="center" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

my main ConstraintLayout

A GridView inside it

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        GridView gridView = findViewById(R.id.grid_view);

        // Instance of ImageAdapter Class
        gridView.setAdapter(new ImageAdapter(this));


   }

}
```

Get the GridView from the layout

Add an Adapter to it to tell it what to show and how

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        GridView gridView = findViewById(R.id.grid_view);

        // Instance of ImageAdapter Class
        gridView.setAdapter(new ImageAdapter(this));


        /**
         * On Click event for Single Gridview Item
         * */
        gridView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View v,
                                    int position, long id) {

                // Sending image id to FullScreenActivity
                Intent i = new Intent(getApplicationContext(), FullImageActivity.class);
                // passing array index
                i.putExtra("id", position);
                startActivity(i);
            }
        });
    }

}
```

Get the GridView from the layout

A listener added on GridView, that reacts to OnItemClick events.

Anonymous Inner class that implements the method onItemClick

When an item is clicked it creates a new intent (to open another activity) and uses the Extra bundle to send info to the other activity (the id of the clicked ImageView)

```
public class FullImageActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_full_image);

        // get intent data
        Intent i = getIntent();

        // Selected image id
        int position = i.getExtras().getInt("id");
        ImageAdapter imageAdapter = new ImageAdapter(this);

        ImageView imageView = (ImageView) findViewById(R.id.full_image_view);
        imageView.setImageResource(imageAdapter.mThumbIds[position]);
    }
}
```

Gets the intent that created it

Reads the Extras bundle

Create an instance of ImageAdapter (we need to to find what image is in location "id)"

Use the location of the image to create an ImageView (this ImageView is defined inside the xml file of the second activity)

activity_full_imagee.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <ImageView android:id="@+id/full_image_view"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"/>

</LinearLayout>
```

More advanced: Reactive Programming

- reactive programming = asynchronous programming = callback methods = observer pattern
- manage **streams** of data
- well-adapted paradigm for asynchronous (UX) programs
- Android library: RxAndroid is an event-based asynchronous programming

Advance Reading

- Next week's topic: Fragments and Navigation
- Fragments: Chapter 29
- Navigation:
  - Principles https://developer.android.com/guide/navigation/navigation-principles
  - Overview https://developer.android.com/guide/navigation
  - https://developer.android.com/guide/navigation/navigation-getting-started

Questions