Lecture 7: More about UI

Dimensions of a View, UI Thread, More about layouts

Jianjun.Chen(Jianjun.Chen@xjtlu.edu.cn)

View Dimension

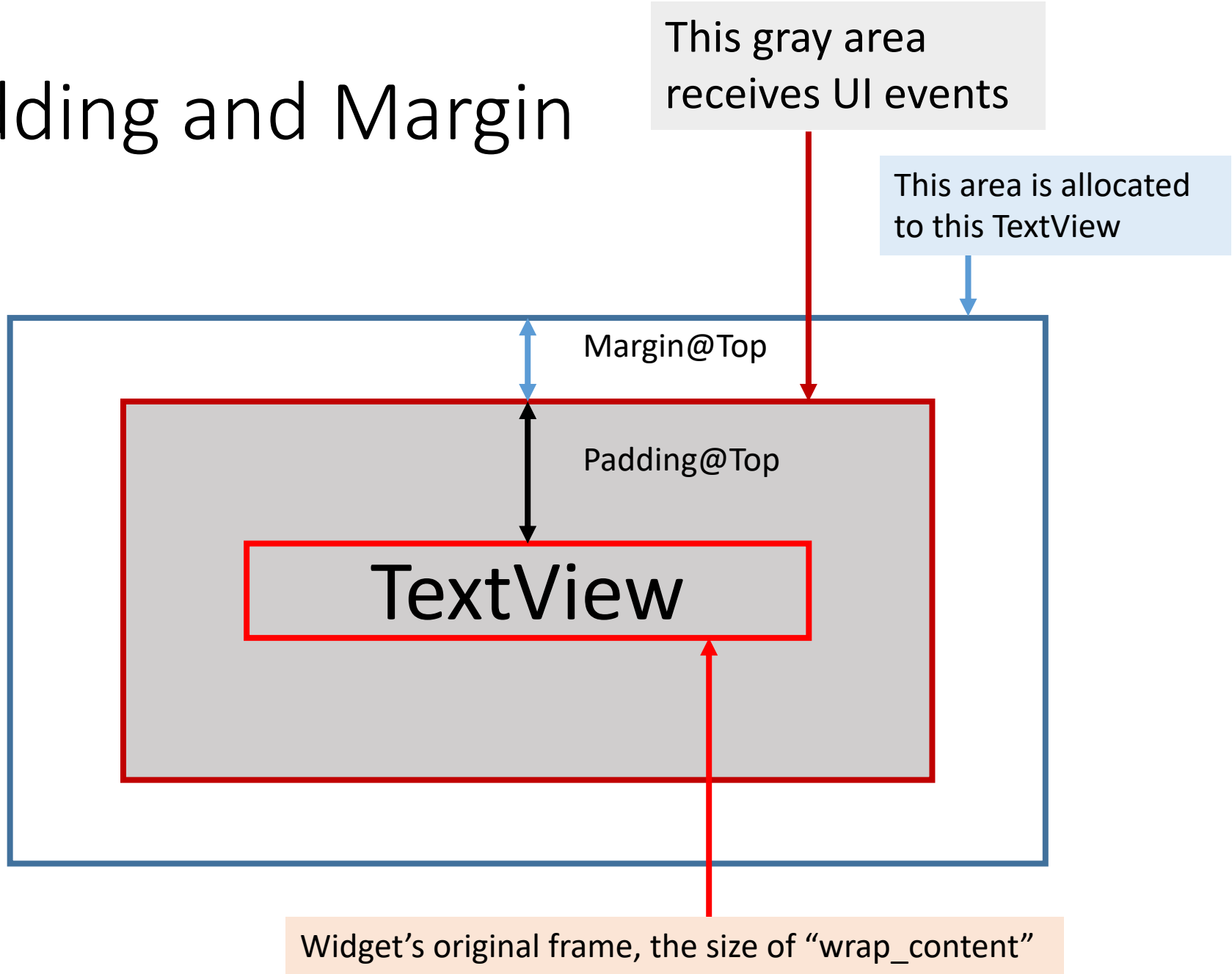
Dimension values, Padding, Margin

Padding and Margin

- We can increase the padding size of a view to make it look bigger.

- We can use margin to create empty space around a view where other views cannot (normally) occupy.

Padding and Margin



Padding and Margin

- The padding area is still a part of a View.
  - Setting the background colour of a View, and the background colour will extend to the padding area.
  - `View.getWidth()` and `View.getHeight()` will report the dimension including the padding area.

- The margin area is not a part of a View.
  - But other views are not supposed to cover this area.

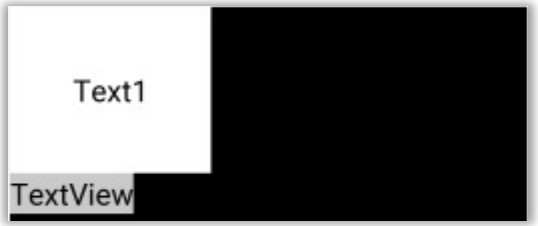
Controlling Padding and Margin

- Padding can be configured using XML attribute:
  - `android:padding="12dp"`
- You can also set paddings for individual directions:
  - `android:paddingBottom="22dp"`

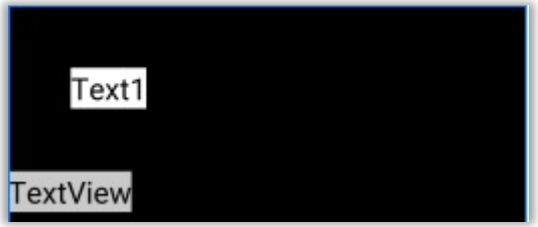
- Margin configuration:
  - `android:layout_margin="20dp"`
  - `android:layout_marginRight="20dp"`

```
<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="0dp"
    android:background="#FFFFFF"
    android:text="Text1"
    android:textColor="#000000"
    android:textSize="24sp" />
```

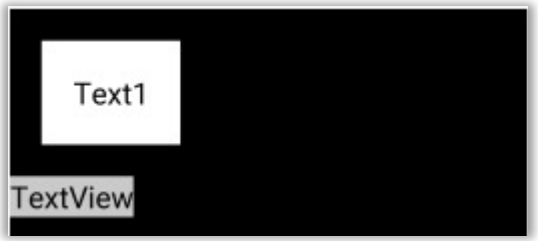

```
android:padding="50dp"
```




```
android:layout_margin="50dp"
```




```
android:padding="25dp"
android:layout_margin="25dp"
```



Units of Measurement

- **Pixel** (px): actual pixels on the screen.
- **Inch** (in): physical size of the screen.
  - 1 Inch = 2.54 centimetres
- **Millimetre** (mm)
- **Points** (pt): 1/72 of an inch.

Units of Measurement

- **Density-independent Pixel** (dp or dip):
  - An abstract unit that is based on the physical density of the screen. These units are relative to a 160-dpi screen.
  - Using dp can ensure proper display of your UI on screens with different dpi settings.

- px = dp * (dpi / 160)
  - If you use a dpi of 240, then 2 dp uses 3 pixels.

- **Scale-independent Pixels** (sp):
  - Similar to dp, but also scaled by the user's font size preference.

Layouts

Layout Parameters, RelativeLayout, ConstraintLayout (not covered)

Layout Parameters

- In Layout XMLs, you can see lines like:

```
android:layout_width="match_parent"
```

```
layout_xxxx
```

- Their java code counterpart is the settings for <u>layout parameters (`LayoutParams`)</u>.
  - Layout parameters are nested classes (defined inside `ViewGroup` subclasses) that implements the class `ViewGroup.LayoutParams`.
- `LayoutParams` are used by views to tell their parents how they want to be laid out.

Base class:

```
android.view.ViewGroup.LayoutParams
```

Subclasses:

```
FrameLayout.LayoutParams,
GridLayout.LayoutParams,
GridLayoutManager.LayoutParams,
LinearLayout.LayoutParams,
LinearLayoutCompat.LayoutParams,
…
```

The Previous Example

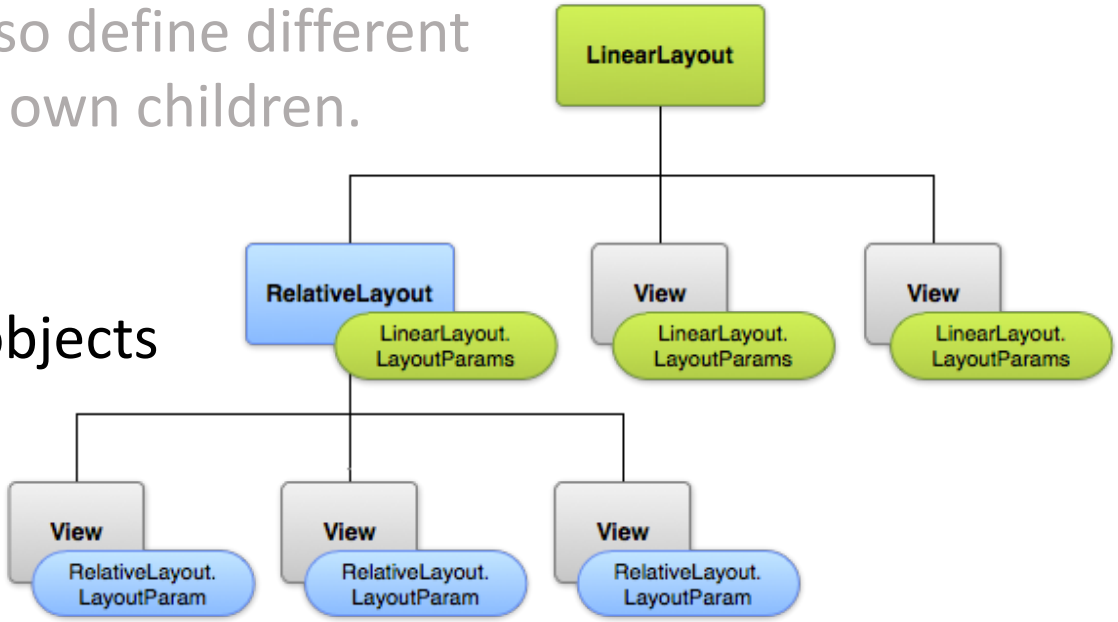
```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //Create params for views---------------
    LinearLayout.LayoutParams params =
            new LinearLayout.LayoutParams(LinearLayout.LayoutParams.FILL_PARENT,
                    LinearLayout.LayoutParams.WRAP_CONTENT);
    //Create a layout---------------
    LinearLayout linearLayout = new LinearLayout( context: this);
    linearLayout.setOrientation(LinearLayout.VERTICAL);

    //----Create a TextView------
    TextView textView = new TextView( context: this);
    textView.setText("This TextView is dynamically created");
    textView.setLayoutParams(params);

    //---Add all elements to the layout
    linearLayout.addView(textView);

    //---Create a layout param for the layout-----------------
    LinearLayout.LayoutParams layoutParams =
            new LinearLayout.LayoutParams(ActionBar.LayoutParams.FILL_PARENT,
                    ActionBar.LayoutParams.WRAP_CONTENT);
    this.addContentView(linearLayout, layoutParams);
}
```

Layout Parameters

"`LayoutParams` are used by views to tell their parents how they want to be laid out."

Thus: Each child element must define `LayoutParams` that are <u>appropriate</u> for its parent, though it may also define different `LayoutParams` for its own children.

To put it simple:
Child `Views` must use objects of `LayoutParams` of its parent.



What is Appropriate?

- Every `LayoutParams` subclass has its own set of settings. Different `LayoutParams` subclasses are **NOT** compatible with different parents.
- For example:
  - `LinearLayout.LayoutParams` has a field called **weight**
  - `GridLayout.layoutParams` instead have **column** and **row**.

- Android will detect and correct when you use a wrong one. But some settings might not be applied properly.

Layout Parameters

- You can specify width and height with exact measurements.
- But the aspect ratios and resolutions of screens of Android phones are rather diverse.
  - A TextView works on one phone might not display properly on another.
- Constants are defined in LayoutParams classes to let Android arrange views automatically:
  - **wrap_content**: tells your view to size itself to the dimensions required by its content.
  - **match_parent**: tells your view to become as big as its parent view group will allow.

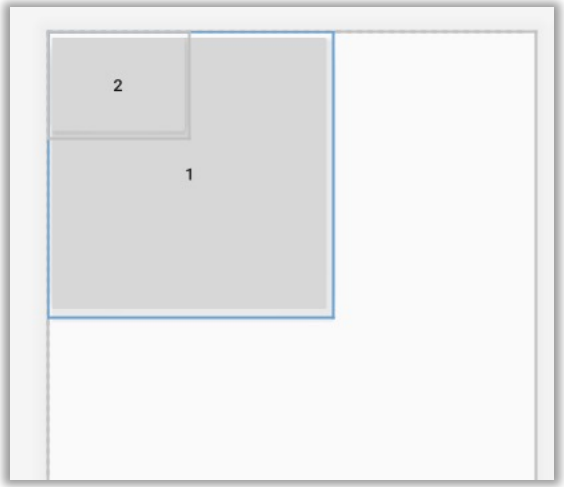
RelativeLayout

- This layout allows child views to be arranged in relative positions.
  - A is on the left side of B.
  - A is above B.

- More efficient and flexible than nested LinearLayout

RelativeLayout: XML Attributes

- A view inside a relative layout has the tendency of sticking to (0, 0), which is the top-left corner.
- If you don’t specify any layout options and add two views.
  - Two views will overlap.
  - The last view added will cover the other.

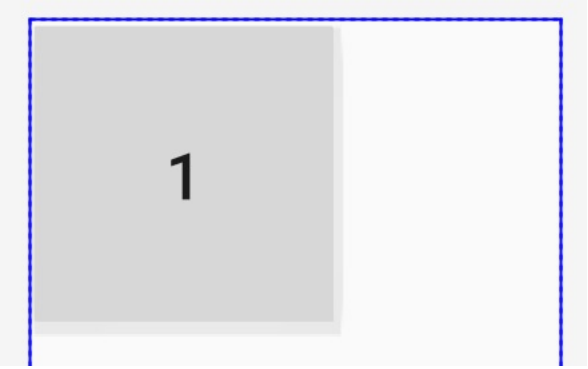


Relative to Parent

```
android:layout_alignParentLeft
    Right|Top|Bottom
```

- This attribute will make a widget stick to the edge of its layout (still inside the layout)

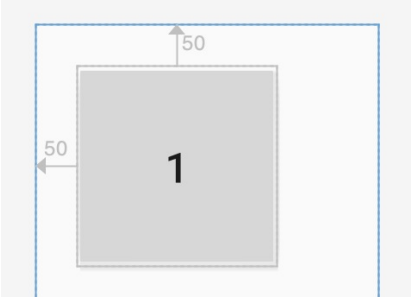
```
android:layout_alignParentLeft="true"
android:layout_alignParentTop="true"
```



Relative to Parent

- You can set margins to give a View some offset from an edge.

```
android:layout_alignParentLeft="true"
android:layout_alignParentTop="true"
android:layout_marginLeft="50dp"
android:layout_marginTop="50dp"
```



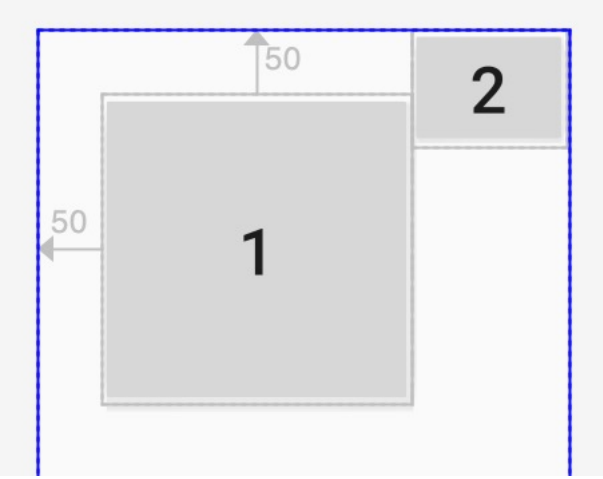
- If you set `layout_alignParentRight` to true, view 1 will be stretched to stick to the right edge of the layout
  - Even if you specify a small width for button 1, it will get stretched.

Relative to a View

`android:layout_above`
`below|toLeftOf|toRightOf`

- This will align a view to an edge of another view.
  - `toLeftOf/toRightOf` can also be replaced by `toStartOf/toEndOf` respectively.

```
<Button
  android:id="@+id/button2"
  android:layout_width="120dp"
  android:layout_height="90dp"
  android:layout_toRightOf="@+id/button1"
  android:text="2"
  android:textSize="50dp" />
```

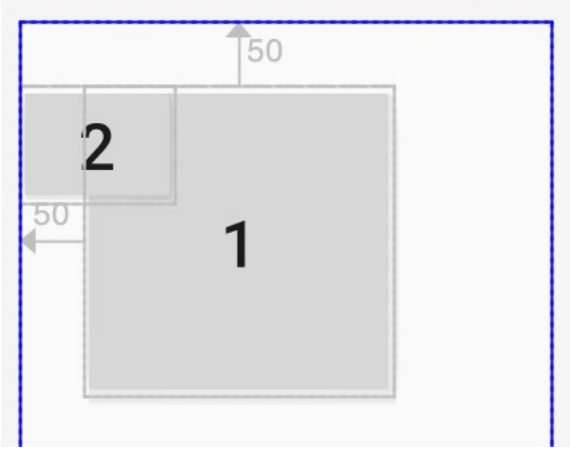


Relative to a View

`android:layout_alignTop`
`Bottom|Left|Right|Baseline`

- This will align the XXX edge of a view to the XXX edge of another view.

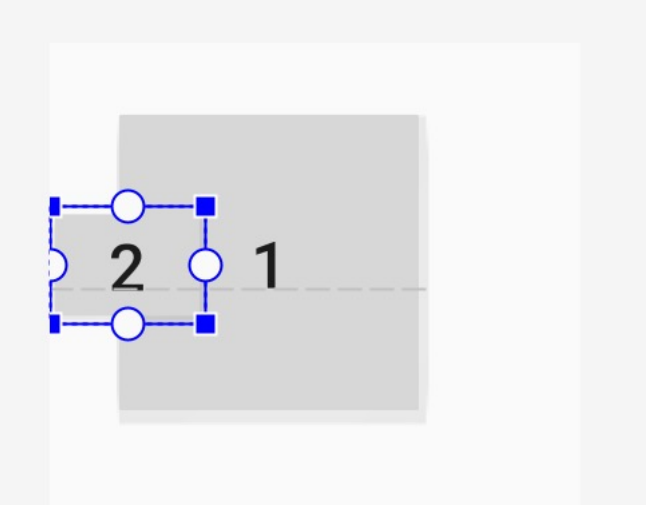
```
<Button
  android:id="@+id/button2"
  android:layout_width="120dp"
  android:layout_height="90dp"
  android:layout_alignTop="@+id/button1"
  android:text="2"
  android:textSize="50dp" />
```



Relative to a View

- Baseline: the invisible line for letters.


```
<Button
  android:id="@+id/button2"
  android:layout_width="120dp"
  android:layout_height="90dp"
  android:layout_alignBaseline="@+id/button1"
  android:text="2"
  android:textSize="50dp" />
```



ConstraintLayout

- Introduced since API level 9.

- Please check the official documentation
- https://developer.android.com/reference/android/support/constraint/ConstraintLayout

Extended Learning

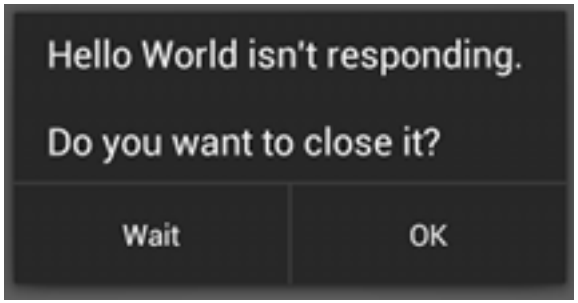
- How to find view’s dimensions before displaying the view
  - https://spotandroid.com/2016/12/21/android-tricks-how-to-find-views-dimensions-before-displaying-the-view/
- View's getWidth() and getHeight() returns 0
  - https://stackoverflow.com/questions/3591784/views-getwidth-and-getheight-returns-0
- Difference between getWidth and getMeasuredWith()
  - https://stackoverflow.com/questions/8657540/what-is-the-difference-between-getwidth-height-and-getmeasuredwidth-height-i

The Android UI Thread

Changing UI concurrently:

runOnUiThread, AsyncTask, View.post()

Android UI Thread



- Android Uses a single thread to draw UI, listen for events and invoke activity life cycle functions of a whole app.

- If this thread gets blocked for too long, Android Not Responsive (ANR) error will be thrown to the user.
  - Block: `Thread.sleep()` or similar situations
  - Bad user experience = a good reason to uninstall.

- To prevent this annoying error, you need to create new threads for long running background work.

Modifying UI from Other Threads

- Now consider this situation:
  - User clicks a play button.
  - The app plays a short piece of music in the background.
  - Then the button's shape need to change to "stopped".

- What is your design?
  - Create a thread to play the music.
  - Wait for the "play" thread? That will block the UI thread.
  - Let the "play" thread finish and then let it change the button?

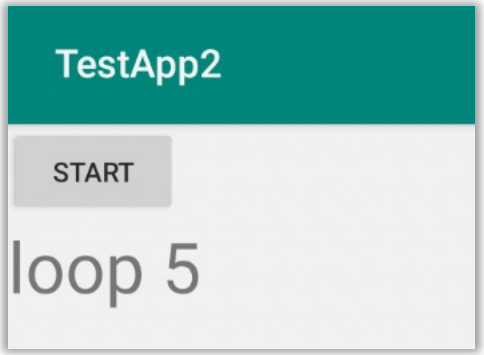
Modifying UI from Other Threads

- Unfortunately, UI (widgets and layouts) can only be changed by UI threads.
  - You can start an activity using non-UI threads though.
- Attempting to use non-UI thread to change UI, `CalledFromWrongThreadException` might be thrown.
  - Using only the UI thread to manage UI can make your app much easier to debug and run more robust.
- There are a few methods to enable UI and non-UI threads to communicate.

Modifying UI from Other Threads

- We are introducing 3 approaches:
  - `Activity.runOnUiThread(Runnable)`
  - `View.post(Runnable)`
  - `AsyncTask`

- Examples scenario: Click start button, then for every 1 second, the content of a `TextView` is changed. Last 5 sec.

Loop 1 -> 2 -> 3 -> 4 -> 

Firstly, the Incorrect Solution

```
Button startButton = findViewById(R.id.startButton);
startButton.setOnClickListener(
new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        for (int i = 1; i <= 5; i++) {
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            TextView label =
                    findViewById(R.id.messageLabel);
            label.setText("loop " + i);
        }
    }
});
```

```java
@Override
public void onClick(View v) {
  new Thread(new Runnable() {
    @Override
    public void run() {
      for(labelVal = 1; labelVal <= 5; labelVal++) {
        try {
          Thread.sleep(1000);
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
        runOnUiThread(new Runnable() {
          @Override
          public void run() {
            TextView label =
                findViewById(R.id.messageLabel);
            label.setText("loop " + labelVal);
          }
        });
      }
    }
  }).start();
}
```

From the UI thread:
Start thread #1

Sleep in Thread #1

Set label in the UI thread

Activity.runOnUiThread()

Alternative Solution?

```
runOnUiThread(new Runnable() {
    @Override
    public void run() {
      try {
        Thread.sleep(1000);
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
      TextView label = findViewById(R.id.messageLabel);
      label.setText("loop " + labelVal);
    }
});
```

Another Wrong Solution

```
runOnUiThread(new Runnable() {
    @Override
    public void run() {
      try {
        Thread.sleep(1000);
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
      TextView label = findViewById(R.id.messageLabel);
      label.setText("loop " + labelVal);
    }
});
```

Still blocks the UI thread

FAIL

```
public void onClick(View v) {
  new Thread(new Runnable() {
    @Override
    public void run() {
      final TextView label = findViewById(R.id.messageLabel);
      for(labelVal = 0; labelVal < 5; labelVal++) {
        try {
          Thread.sleep(1000);
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
        label.post(new Runnable() {
          @Override
          public void run() {
            label.setText("loop " + labelVal);
          }
        });
      }
    }
  }).start();
}
```

UI Thread

View.post(Runnable)

Method 3: AsyncTask

- AsyncTask is a generic class that has four key functions.

- Three functions that run on the UI thread:
  - `onPreExecute()`
  - `onPostExecute()`
  - `onProgressUpdate()`

- And a function that is carried out in another tread:
  - `doInBackground()`

AsyncTask: the Work Flow

- Create an object of `AsyncTask`, and call `execute()`

```
AsyncTask<Params, Progress, Result>
```

- `onPreExecute()` will be run on the UI thread first.
- `doInBackground(Params... params)` will then be run on a separate thread.
  - You can call `onProgressUpdate(Progress... values)` during this period, all logic will be run on the UI thread.
- `onPostExecute(Result result)` will be done on the UI thread.

Solution Using AsyncTask

To implement our task using `AsyncTask`:

1. Create a subclass of `AsyncTask`, preferably inside our `Activity` class so that we can access important functions like `findViewByID()`.

2. Override `doInBackground()` and `onPostExecute()`.

3. Create an object of our `AsyncTask` and use it inside `onClick()` function of the “start” button

```
public class MainActivity extends AppCompatActivity {

  class SetLabelTask extends
        AsyncTask<Integer, Void, Integer> {
     @Override
     protected Integer doInBackground(Integer... integers) {
        try {
           Thread.sleep(1000);
        } catch (InterruptedException e) {
           e.printStackTrace();
        }
        return integers[0];
     }
     @Override
     protected  void onPostExecute(Integer i) {
        final TextView label =
                        findViewById(R.id.messageLabel);
        label.setText("Loop " + i);
     }
  }
……
```

The “loop i” integer

AsyncTask: Create Class

```
AsyncTask<Params, Progress, Result>

protected Result doInBackground(Params... params)
{
    …
    return result;
}

protected  void onPostExecute(Result i) {
    …
}
```

Non-UI Thread

UI Thread

AsyncTask: Create Class

```
public void onClick(View v) {
    for (int i = 0; i < 5; i++) {
        new SetLabelTask().execute(i + 1);
    }
}
```

```
doInBackground(Params... params)
```

```
AsyncTask<Params, Progress, Result>
```

You may cancel an `AsyncTask` at any time, in any thread, using `AsyncTask.cancel()` function

AsyncTask: implement onClick()

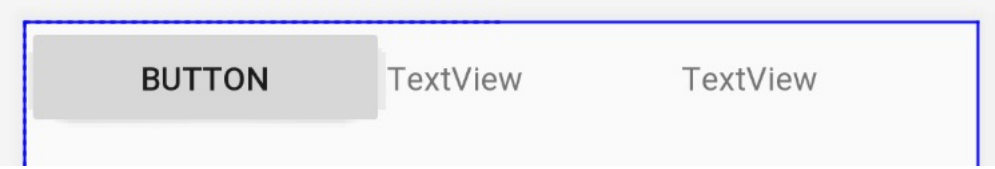
Lab Tasks

- Task 1: The AsyncTask example in the slides is not ideal. Redesign this app using `AsyncTask.doInBackground()` and `AsyncTask.onProgressUpdate()`.

- Task 2: Redesign your activity layouts from the last lecture using relative layout and constraint layout.
  - Do some experiments, drag you widgets around and see the changes reflected in the layout XML.

Lab Tasks

- Task 3: Create a simple layout like below.



- When the app starts, the two TextViews' text colour alternates between blue and red at the frequency of 1 change per second.
- By clicking the button, the colour will stay unchanged for 5 seconds and then start again.

Lab Tasks

- Text colour alternation:

Hello world

Hello world

Hello world

Hello world

Hello world

- Do not block the UI thread!