Android Developer Fundamentals V2

Build your first app

Lesson 1

1.1 Your first Android app

Contents

- Android Studio
- Creating "Hello World" app in Android Studio
- Basic app development workflow with Android Studio
- Running apps on virtual and physical devices

Prerequisites

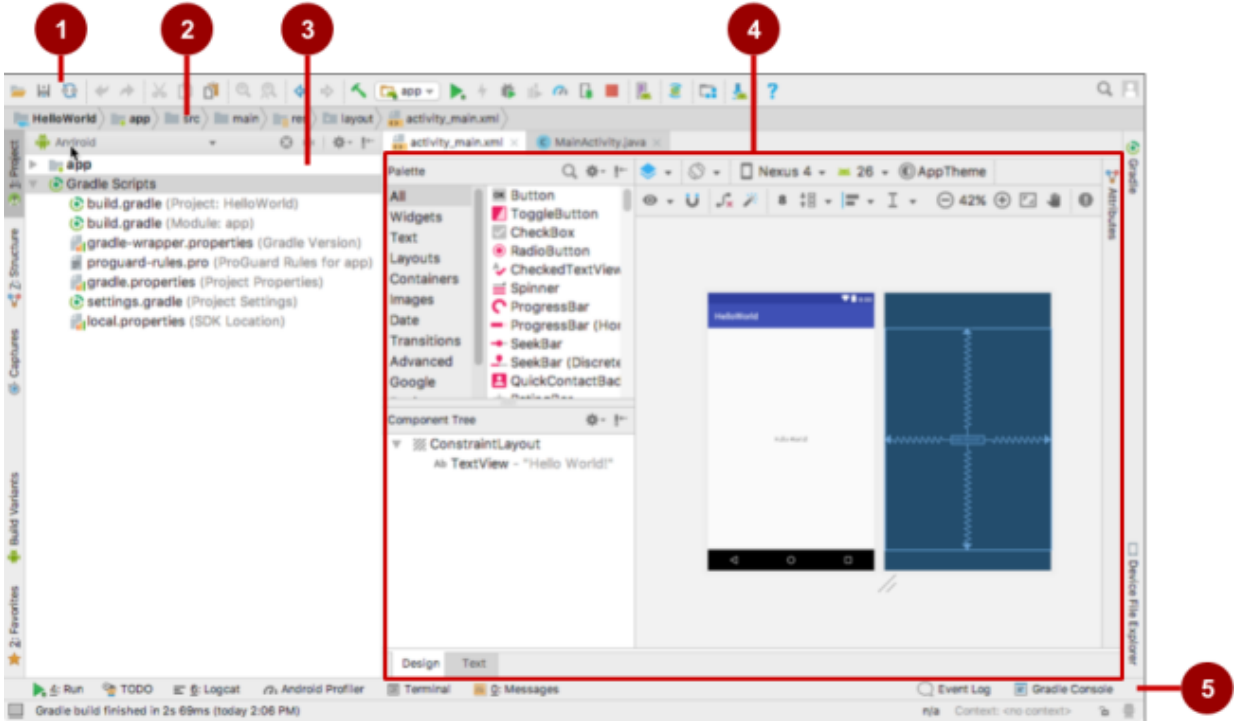
- Java Programming Language
- Object-oriented programming
- XML - properties / attributes
- Using an IDE for development and debugging

Android Studio

What is Android Studio?

- Android integrated development environment (IDE)
- Project and Activity templates
- Layout editor
- Testing tools
- Gradle-based build
- Log console and debugger
- Emulators

Android Studio interface



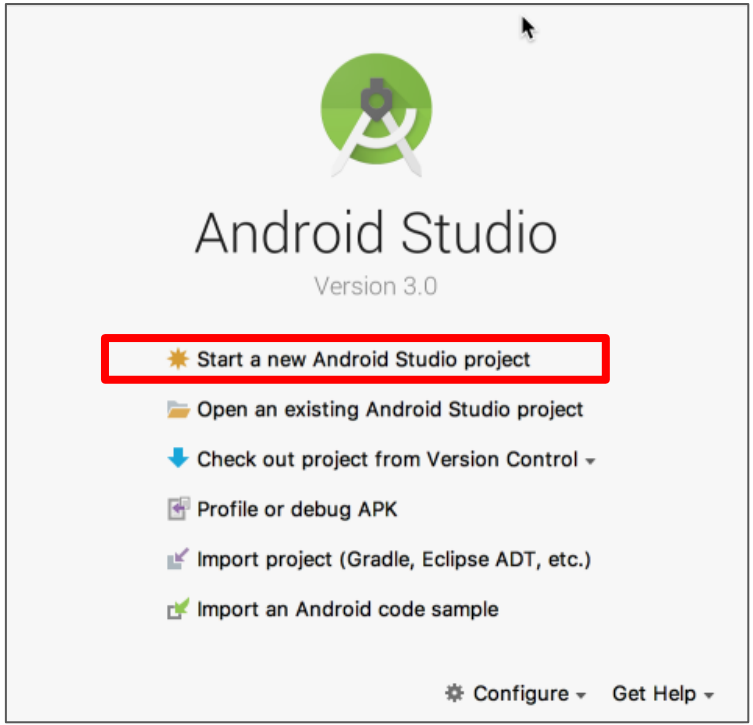
1. Toolbar
2. Navigation bar
3. Project pane
4. Editor
5. Tabs for other panes

Installation Overview

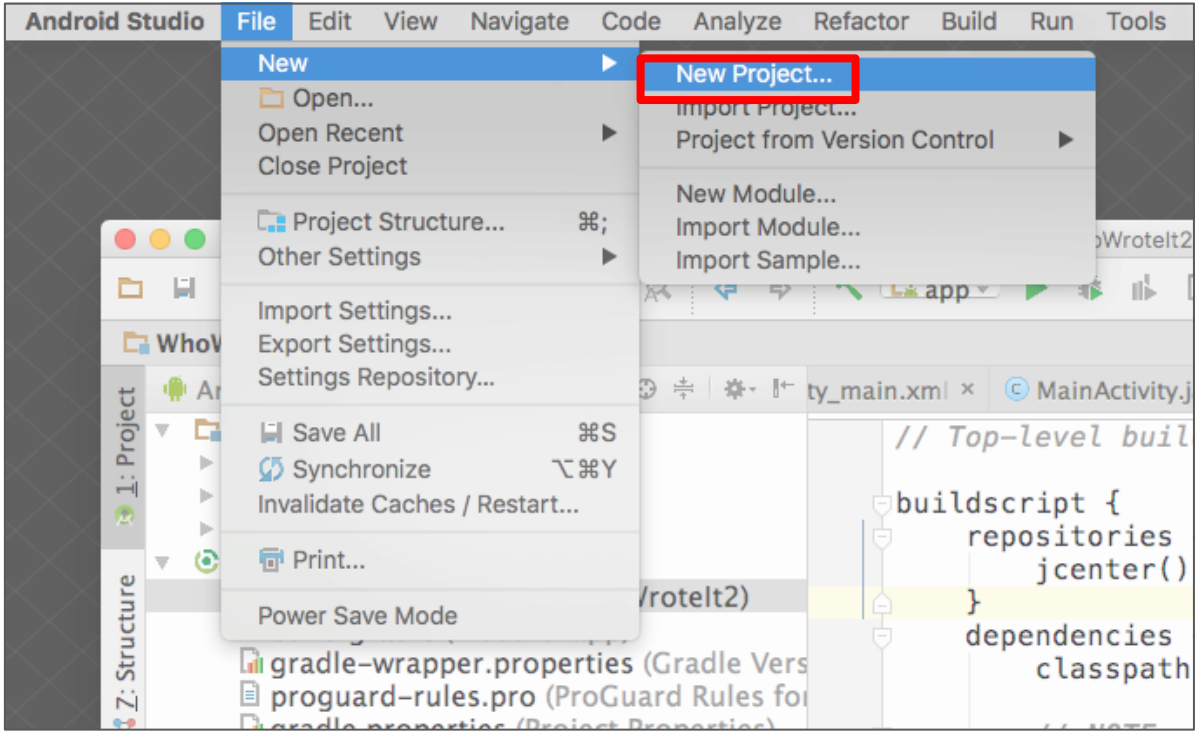
- Mac, Windows, or Linux
- Download and install Android Studio from https://developer.android.com/studio/
- See 1.1 P: Android Studio and Hello World

Creating your first Android app

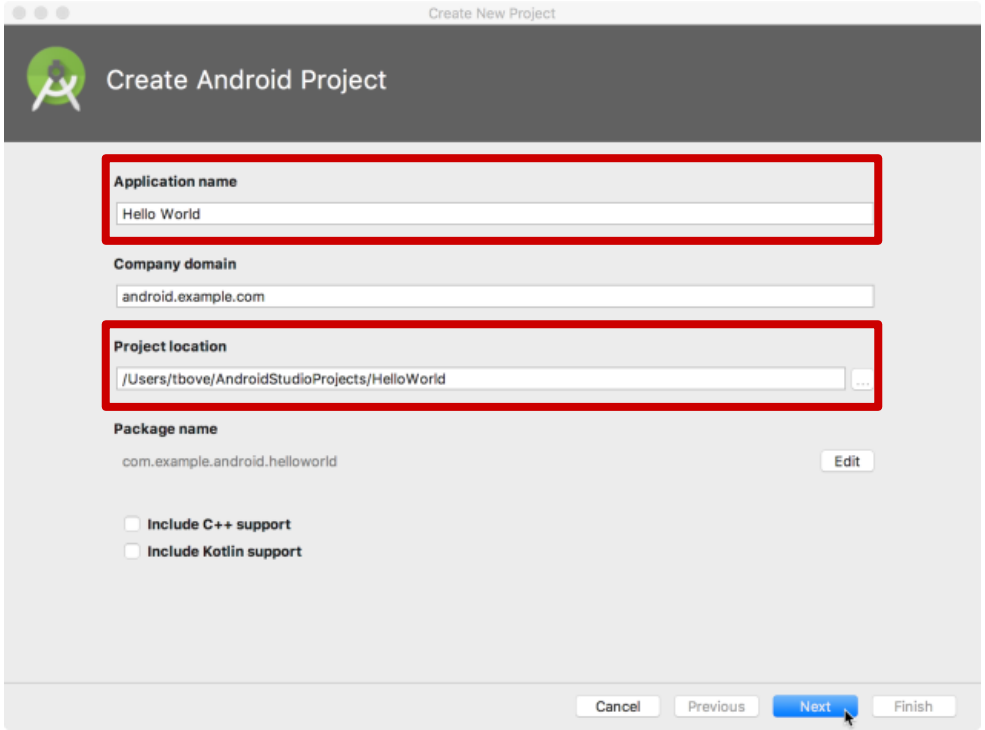
Start Android Studio



Create a project inside Android Studio



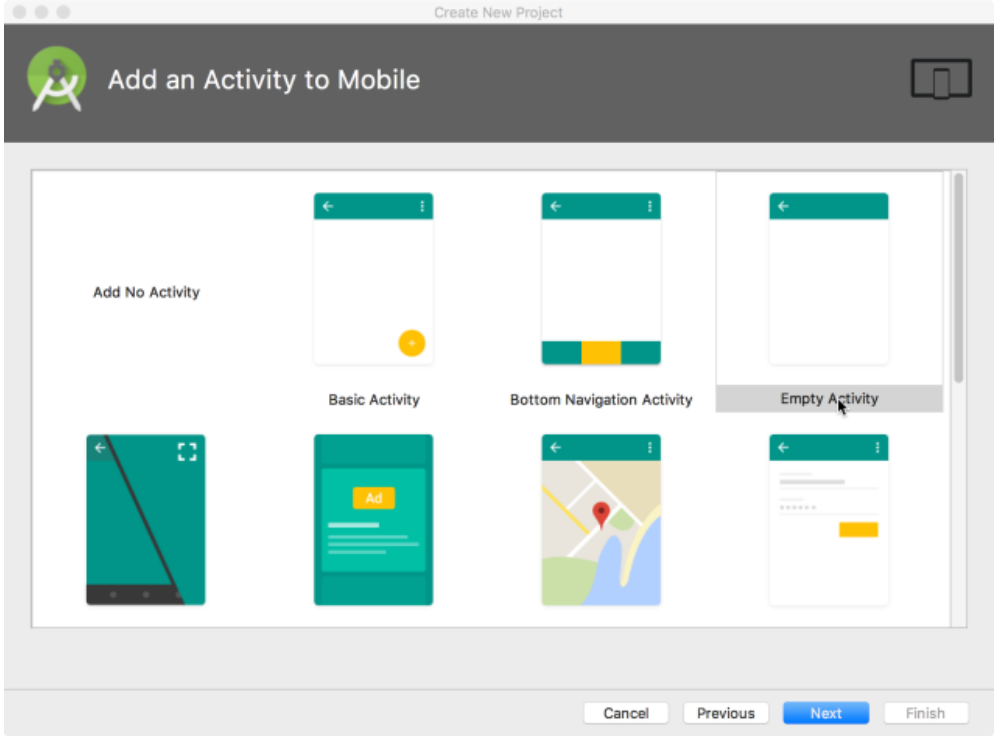
Name your app



Pick activity template

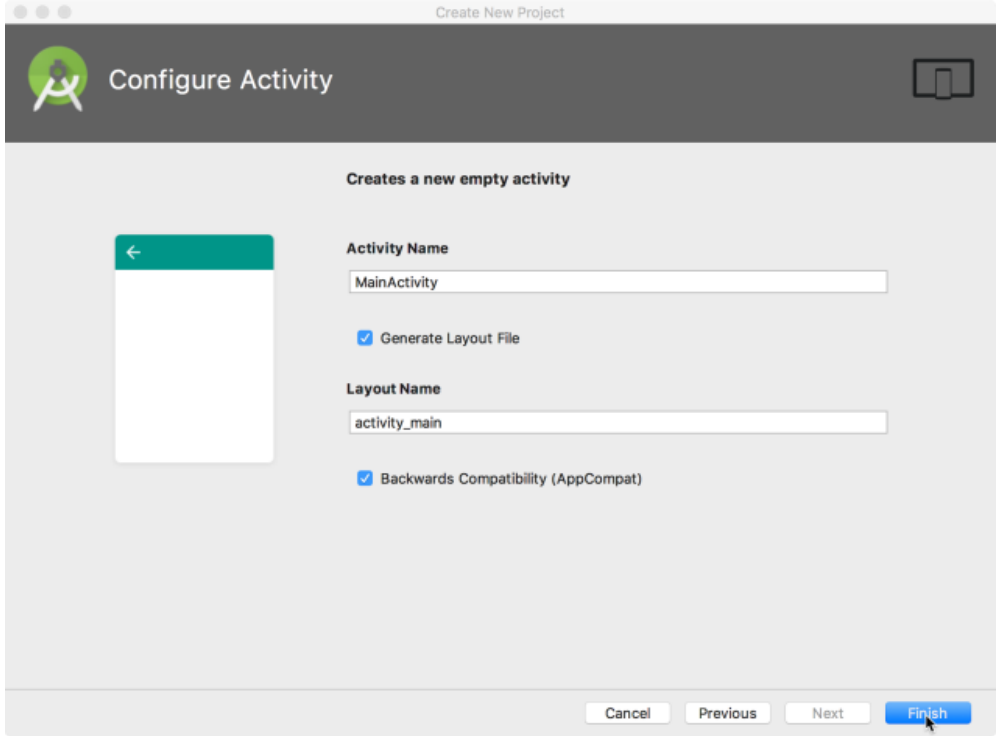
Choose templates for common activities, such as maps or navigation drawers.

Pick Empty Activity or Basic Activity for simple and custom activities.



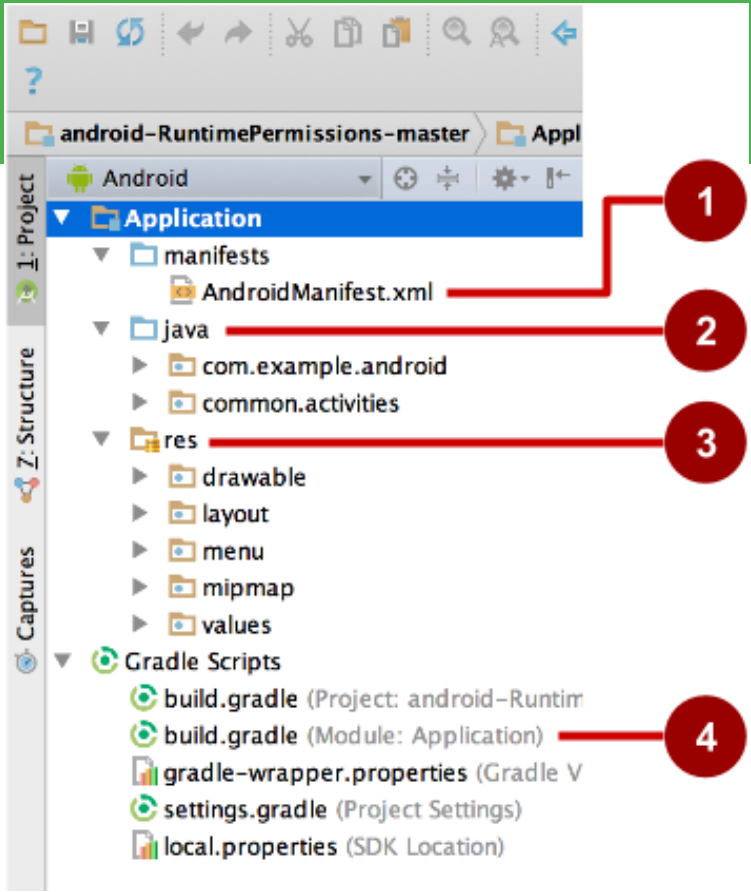
Name your activity

- Good practice:
  - Name main activity `MainActivity`
  - Name layout `activity_main`
- Use AppCompat
- Generating layout file is convenient



Project folders

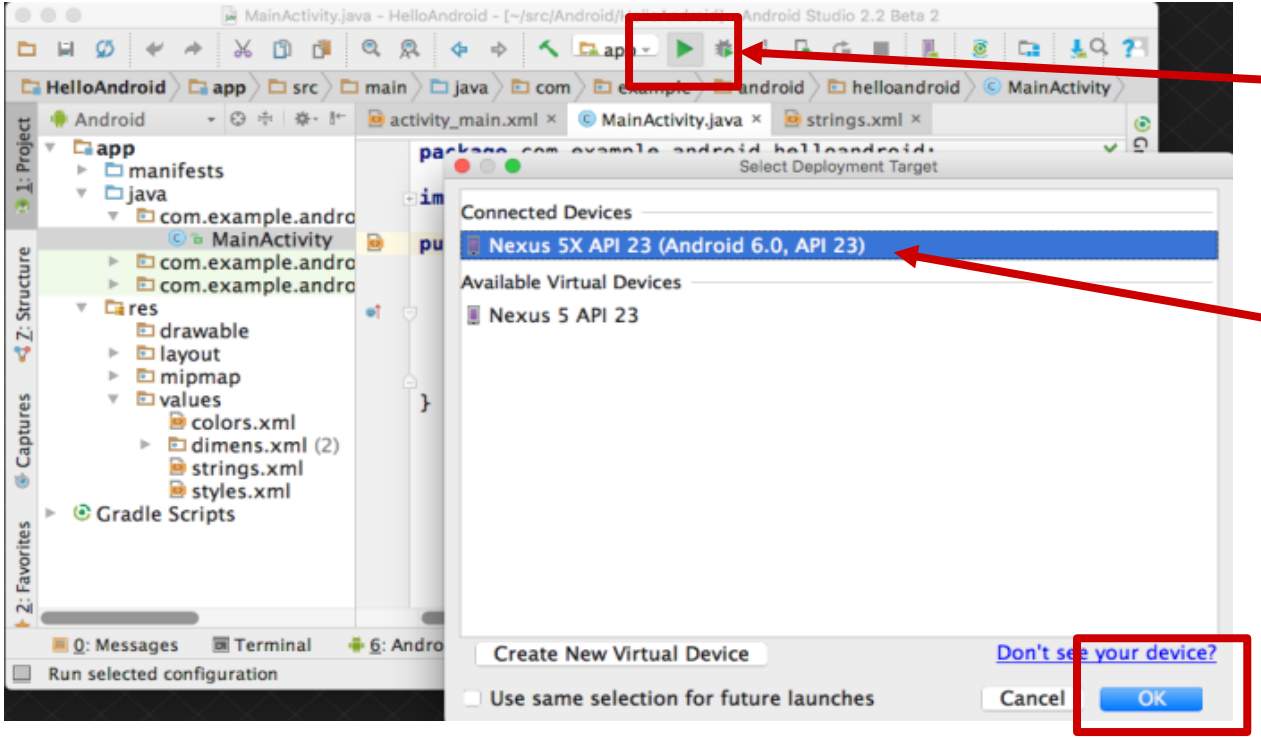
1. **manifests**—Android Manifest file - description of app read by the Android runtime
2. **java**—Java source code packages
3. **res**—Resources (XML) - layout, strings, images, dimensions, colors...
4. **build.gradle**—Gradle build files



Gradle build system

- Modern build subsystem in Android Studio
- Three build.gradle:
  - project
  - module
  - settings
- Typically not necessary to know low-level Gradle details
- Learn more about gradle at https://gradle.org/

Run your app



1. Run

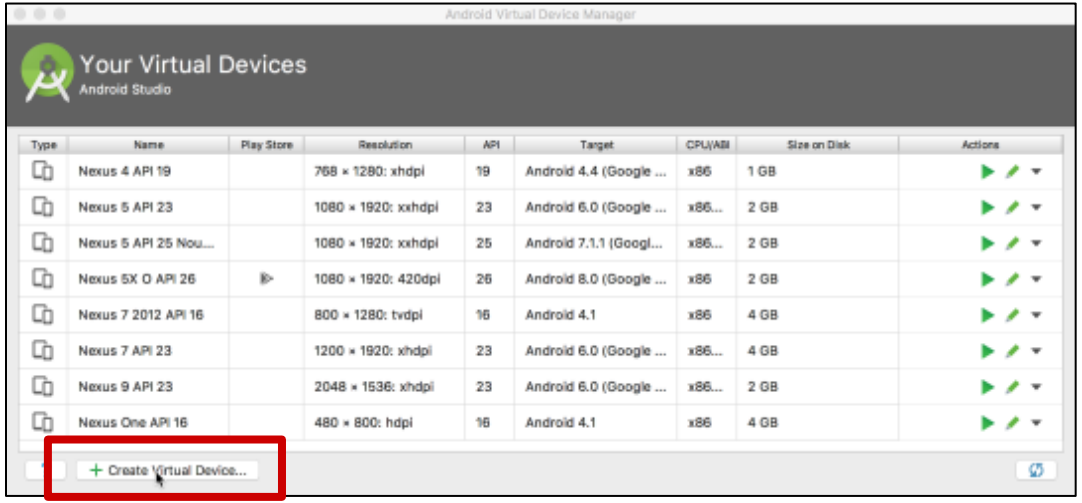
2. Select virtual or physical device

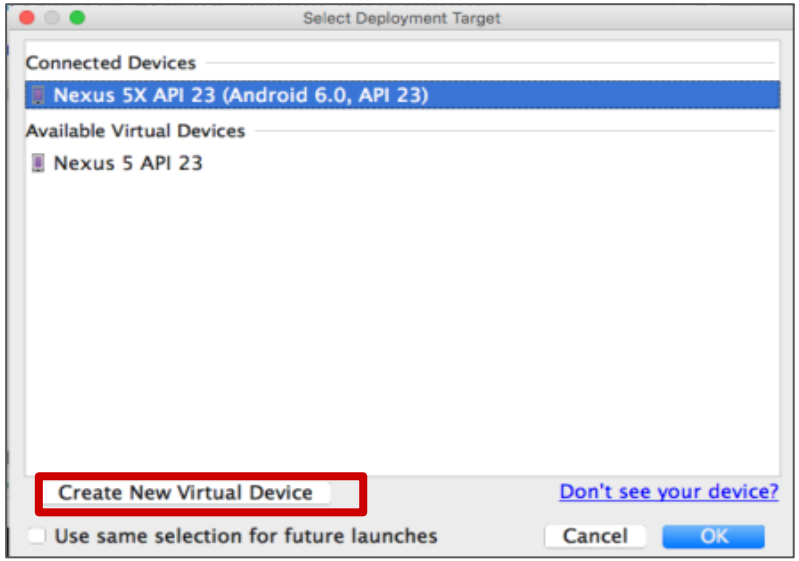
3. OK

Create a virtual device

Use emulators to test app on different versions of Android and form factors.

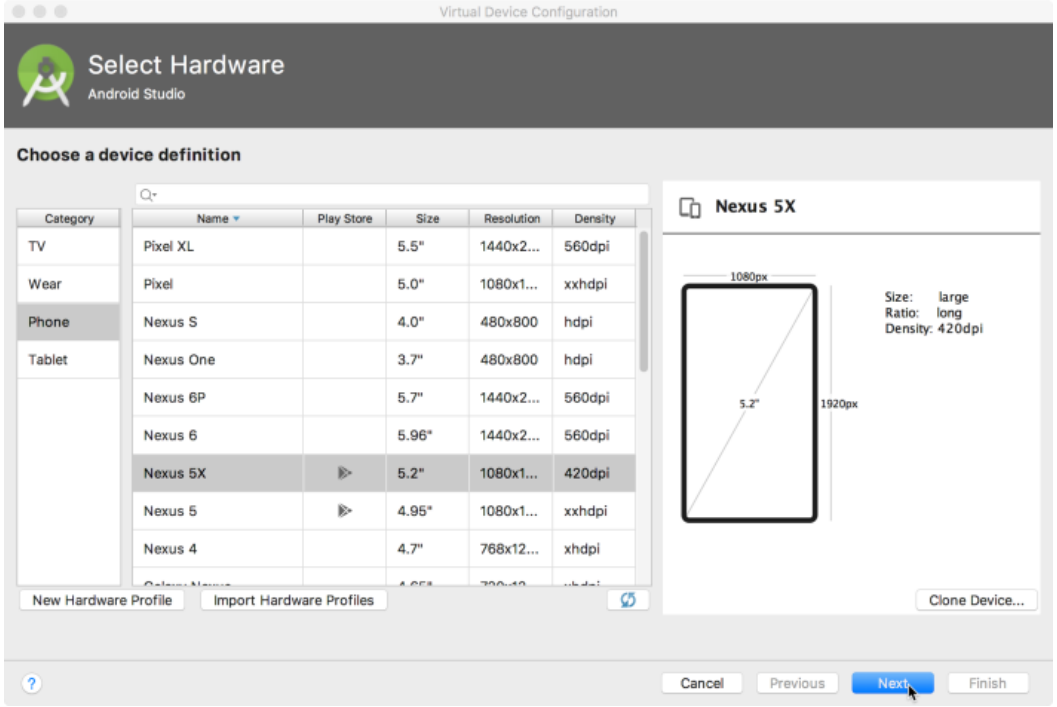
Tools > Android > AVD Manager or:



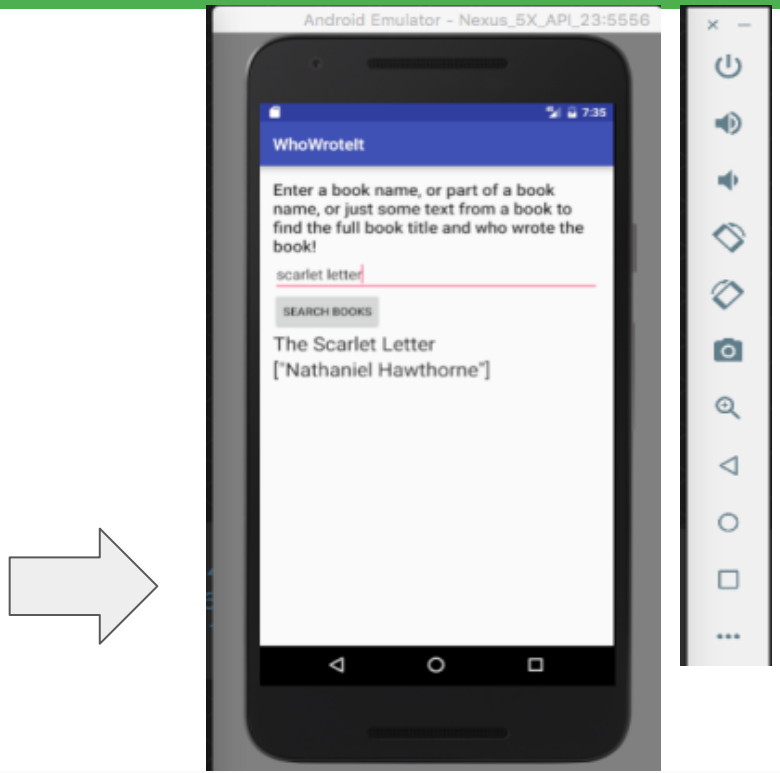


Configure virtual device

1. Choose hardware
2. Select Android version
3. Finalize



Run on a virtual device



Run on a physical device

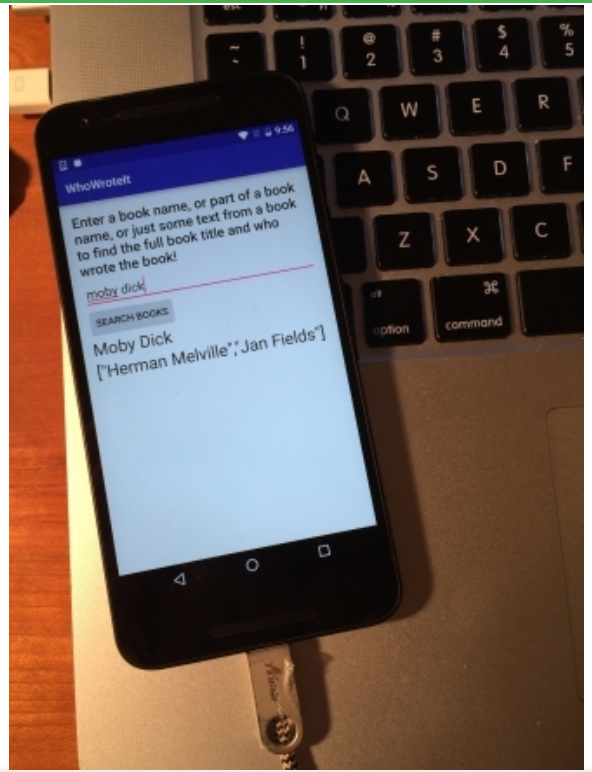
1. Turn on Developer Options:
   a. **Settings > About phone**
   b. Tap **Build number** seven times
2. Turn on USB Debugging
   a. **Settings > Developer Options > USB Debugging**
3. Connect phone to computer with cable

Windows/Linux additional setup:

- Using Hardware Devices

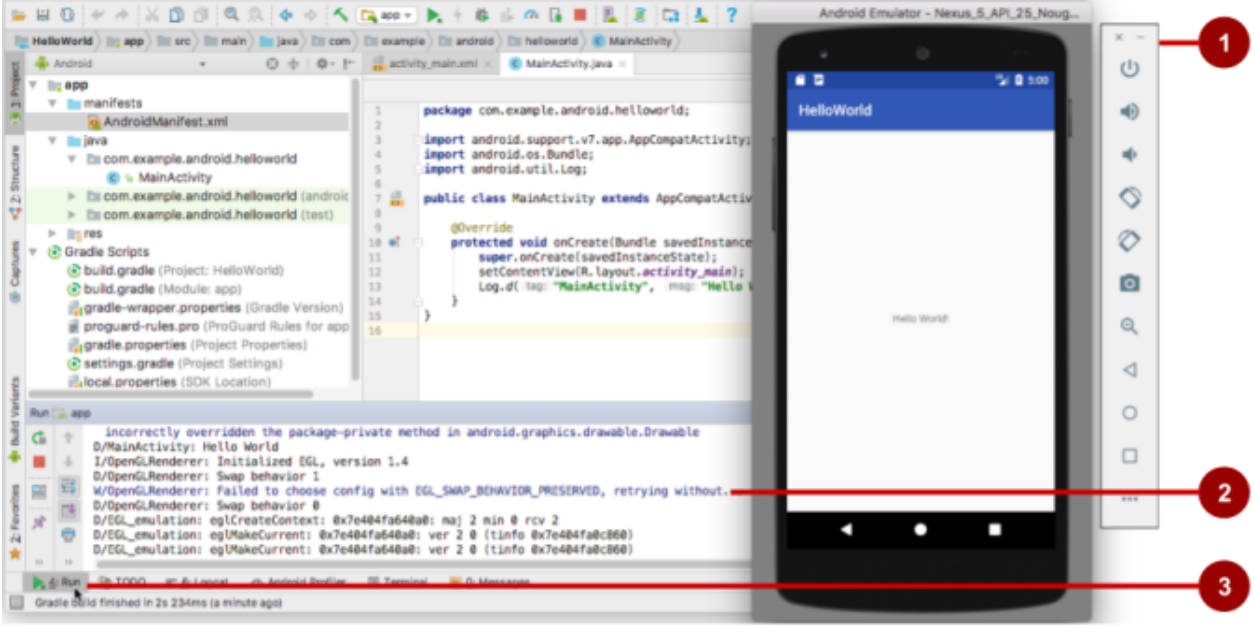
Windows drivers:

- OEM USB Drivers



Get feedback as your app runs

1. Emulator running the app
2. Run pane
3. **Run** tab to open or close the Run pane

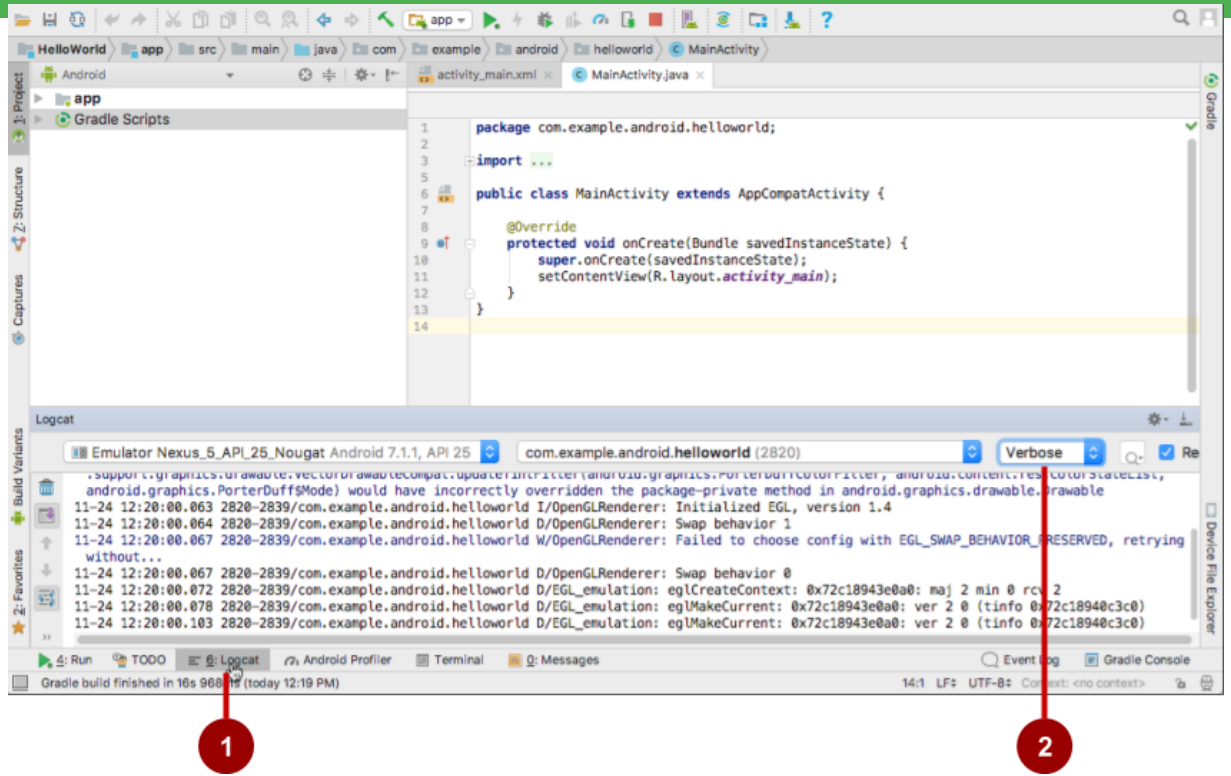


Adding logging to your app

- As the app runs, the **Logcat** pane shows information
- Add logging statements to your app that will show up in the Logcat pane
- Set filters in **Logcat** pane to see what's important to you
- Search using tags

The Logcat pane

1. **Logcat** tab to show Logcat pane
2. Log level menu



Logging statement

```
import android.util.Log;

// Use class name as tag
private static final String TAG =
    MainActivity.class.getSimpleName();

// Show message in Android Monitor, logcat pane
// Log.<log-level>(TAG, "Message");
Log.d(TAG, “Creating the URI…”);
```

Learn more

- Meet Android Studio
- Official Android documentation at developer.android.com
- Create and Manage Virtual Devices
- Supporting Different Platform Versions
- Supporting Multiple Screens

Learn even more

- Gradle Wikipedia page
- Google Java Programming Language style guide
- Find answers at Stackoverflow.com

What's Next?

- Concept Chapter: 1.1 Your first Android app
- Practical: 1.1 Android Studio and Hello World

END

Android Developer Fundamentals V2

Build your first app

Lesson 1

1.2 Layouts and resources for the UI

Contents

- Views, view groups, and view hierarchy
- The layout editor and `ConstraintLayout`
- Event handling
- Resources and measurements

Views

Everything you see is a view

If you look at your mobile device, every user interface element that you see is a **`View`**.

What is a view?

View subclasses are basic user interface building blocks

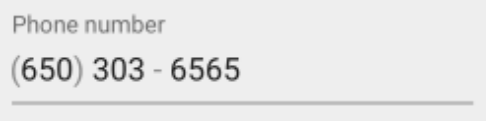
- Display text (TextView class), edit text (EditText class)
- Buttons (Button class), menus, other controls
- Scrollable (ScrollView, RecyclerView)
- Show images (ImageView)
- Group views (ConstraintLayout and LinearLayout)

Examples of view subclasses

Button

CheckBox

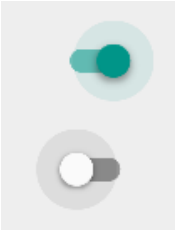
EditText



RadioButton

Slider

Switch



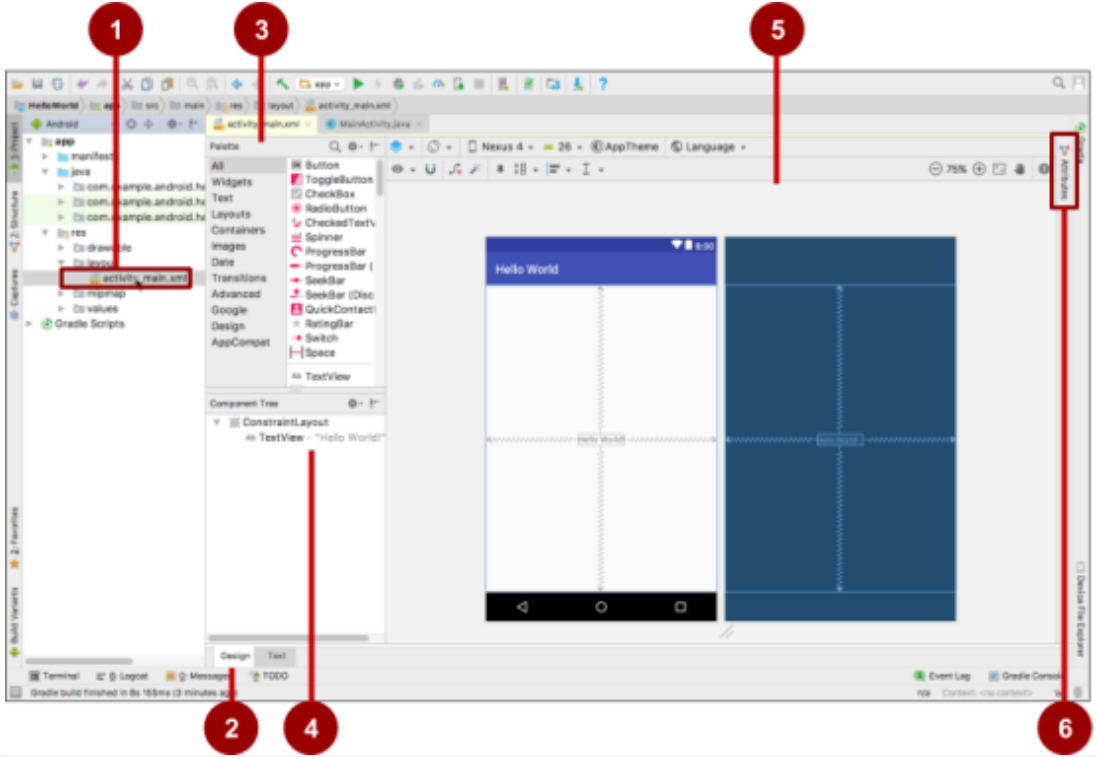
View attributes

- Color, dimensions, positioning
- May have focus (e.g., selected to receive user input)
- May be interactive (respond to user clicks)
- May be visible or not
- Relationships to other views

Create views and layouts

- Android Studio layout editor: visual representation of XML
- XML editor
- Java code

Android Studio layout editor



1. XML layout file
2. **Design** and **Text** tabs
3. **Palette** pane
4. **Component Tree**
5. Design and blueprint panes
6. **Attributes** tab

View defined in XML

```
<TextView
        android:id="@+id/show_count"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@color/myBackgroundColor"
        android:text="@string/count_initial_value"
        android:textColor="@color/colorPrimary"
        android:textSize="@dimen/count_text_size"
        android:textStyle="bold"
/>
```

View attributes in XML

```
android:<property_name>="<property_value>"
Example: android:layout_width="match_parent"

android:<property_name>="@<resource_type>/resource_id"
Example: android:text="@string/button_label_next"

android:<property_name>="@+id/view_id"
Example: android:id="@+id/show_count"
```

Create View in Java code

In an `Activity`:

context

```
TextView myText = new TextView(this);
myText.setText("Display this text!");
```

What is the context?

- `Context` is an interface to global information about an application environment
- Get the context:

```
Context context = getApplicationContext();
```

- An `Activity` is its own context:

```
TextView myText = new TextView(this);
```

Custom views

- Over 100 (!) different types of views available from the Android system, all children of the `View` class
- If necessary, create custom views by subclassing existing views or the `View` class

ViewGroup and View hierarchy

ViewGroup contains "child" views

- `ConstraintLayout`: Positions UI elements using constraint connections to other elements and to the layout edges
- `ScrollView`: Contains one element and enables scrolling
- `RecyclerView`: Contains a list of elements and enables scrolling by adding and removing elements dynamically

ViewGroups for layouts

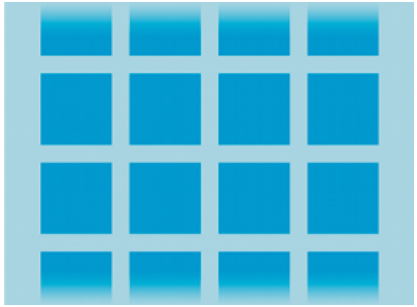
Layouts

- are specific types of `ViewGroups` (subclasses of `ViewGroup`)
- contain child views
- can be in a row, column, grid, table, absolute

Common Layout Classes

LinearLayout

ConstraintLayout



GridLayout

TableLayout

Common Layout Classes

- `ConstraintLayout`: Connect views with constraints
- `LinearLayout`: Horizontal or vertical row
- `RelativeLayout`: Child views relative to each other
- `TableLayout`: Rows and columns
- `FrameLayout`: Shows one child of a stack of children

Layout created in XML

```
<LinearLayout
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent">
    <Button
        ... />
    <TextView
        ... />
    <Button
        ... />
</LinearLayout
```

Event Handling

Events

Something that happens

- In UI: Click, tap, drag
- Device: DetectedActivity such as walking, driving, tilting
- Events are "noticed" by the Android system

Event Handlers

Methods that do something in response to a click

- A method, called an **event handler**, is triggered by a specific event and does something in response to the event

Attach in XML and implement in Java

Attach handler to view in XML layout:

```
android:onClick="showToast"
```

Implement handler in Java activity:

```
public void showToast(View view) {
  String msg = "Hello Toast!";
  Toast toast = Toast.makeText(
        this, msg, duration);
  toast.show();
}
}
```

Alternative: Set click handler in Java

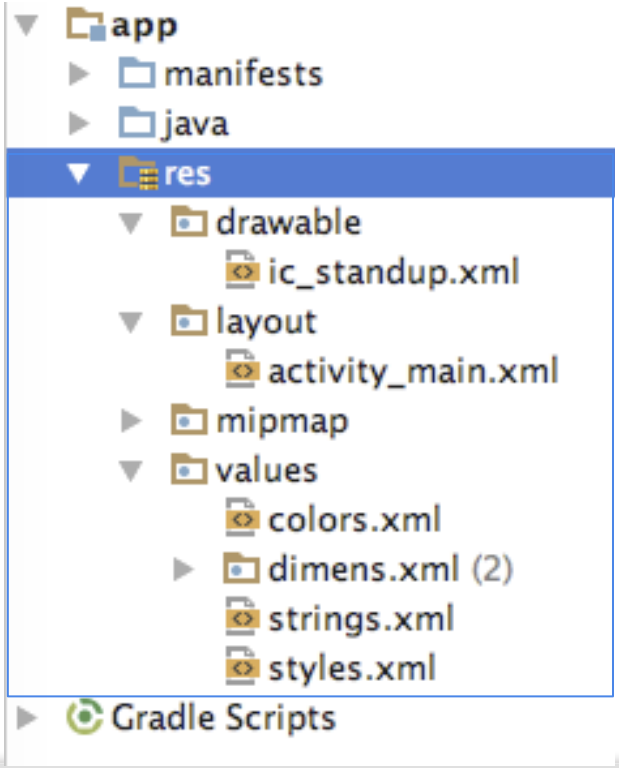
```
final Button button = (Button) findViewById(R.id.button_id);
button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        String msg = "Hello Toast!";
        Toast toast = Toast.makeText(this, msg, duration);
        toast.show();
        }
    });
```

Resources and measurements

Resources

- Separate static data from code in your layouts.
- Strings, dimensions, images, menu text, colors, styles
- Useful for localization

Where are the resources in your project?



resources and resource files stored in **res** folder

Refer to resources in code

- Layout:

  `R.layout.activity_main`

  `setContentView(R.layout.activity_main);`

- View:

  `R.id.recyclerview`

  `rv = (RecyclerView) findViewById(R.id.recyclerview);`

- String:

  In Java: `R.string.title`

  In XML: `android:text="@string/title"`

Measurements

- Density-independent Pixels (`dp`): for Views
- Scale-independent Pixels (`sp`): for text

Don't use device-dependent or density-dependent units:

- ~~Actual Pixels (px)~~
- ~~Actual Measurement (`in`, `mm`)~~
- ~~Points - typography 1/72 inch (`pt`)~~

Learn more

Learn more

Views:

- `View` class documentation
- device independent pixels
- `Button` class documentation
- `TextView` class documentation

Layouts:

- developer.android.com Layouts
- Common Layout Objects

Learn even more

Resources:

- Android resources
- `Color` class definition
- `R.color` resources
- Supporting Different Densities
- Color Hex Color Codes

Other:

- Android Studio documentation
- Image Asset Studio
- UI Overview
- Vocabulary words and concepts glossary
- Model-View-Presenter (MVP) architecture pattern
- Architectural patterns

What's Next?

- Concept Chapter: 1.2 Layouts and resources for the UI
- Practicals:
  - 1.2A : Your first interactive UI
  - 1.2B : The layout editor

END

Android Developer Fundamentals V2

Build your first app

Lesson 1

1.3 Text and scrolling views

Contents

- TextView
- ScrollView

TextView

TextView for text

- `TextView` is `View` subclass for single and multi-line text
- `EditText` is `TextView` subclass with editable text
- Controlled with layout attributes
- Set text:
  - Statically from string resource in XML
  - Dynamically from Java code and any source

Creating TextView in XML

```
<TextView android:id="@+id/textview"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/my_story"/>
```

Common TextView attributes

`android:text`—text to display

`android:textColor`—color of text

`android:textAppearance`—predefined style or theme

`android:textSize`—text size in `sp`

`android:textStyle`—`normal`, `bold`, `italic`, or `bold|italic`

`android:typeface`—`normal`, `sans`, `serif`, or `monospace`

`android:lineSpacingExtra`—extra space between lines in `sp`

Creating TextView in Java code

```
TextView myTextview = new TextView(this);
myTextView.setWidth(LayoutParams.MATCH_PARENT);
myTextView.setHeight(LayoutParams.WRAP_CONTENT);
myTextView.setMinLines(3);
myTextView.setText(R.string.my_story);
myTextView.append(userComment);
```

ScrollView

What about large amounts of text?

- News stories, articles, etc…
- To scroll a `TextView`, embed it in a `ScrollView`
- Only *one* `View` element (usually `TextView`) allowed in a `ScrollView`
- To scroll multiple elements, use one `ViewGroup` (such as `LinearLayout`) within the `ScrollView`

ScrollView for scrolling content

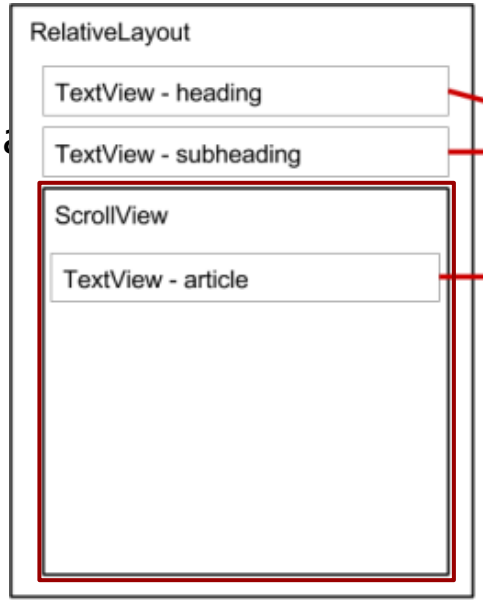
- `ScrollView` is a subclass of `FrameLayout`
- Holds all content in memory
- Not good for long texts, complex layouts
- Do not nest multiple scrolling views
- Use `HorizontalScrollView` for horizontal scrolling
- Use a `RecyclerView` for lists

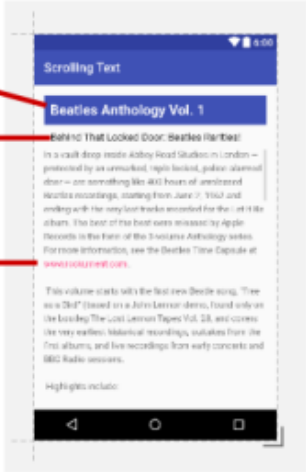
ScrollView layout with one TextView

```
<ScrollView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_below="@id/article_subhea

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    .../>

</ScrollView>
```



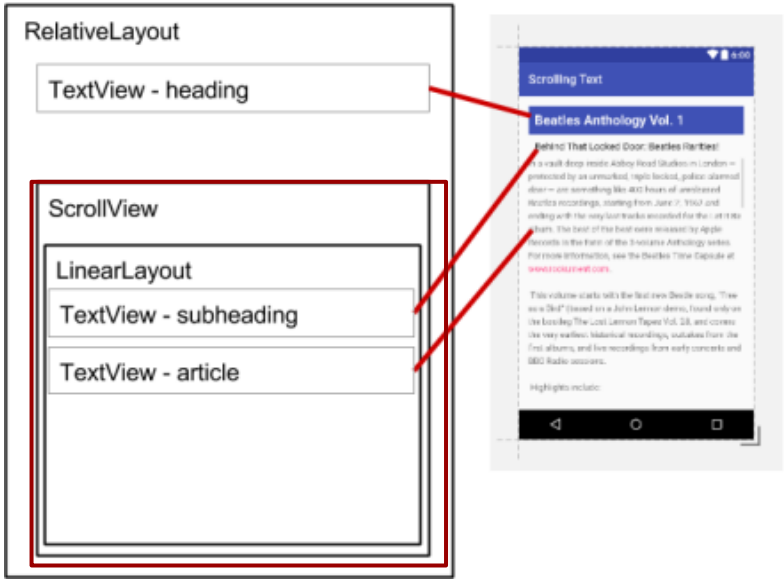


ScrollView layout with a view group

```
<ScrollView ...
   <LinearLayout
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:orientation="vertical">

       <TextView
           android:id="@+id/article_subheading"
           .../>

       <TextView
           android:id="@+id/article" ... />
   </LinearLayout>
</ScrollView>
```



ScrollView with image and button

```
<ScrollView...>
    <LinearLayout...>
        <ImageView.../>
        <Button.../>
        <TextView.../>
    </LinearLayout>
</ScrollView>
```

One child of ScrollView which can be a layout

Children of the layout

Learn more

Developer Documentation:

- `TextView`
- `ScrollView` and `HorizontalScrollView`
- String Resources

Other:

- Android Developers Blog: Linkify your Text!
- Codepath: Working with a TextView

What's Next?

- Concept Chapter: 1.3 Text and scrolling views
- Practical: 1.3 Text and scrolling views

END