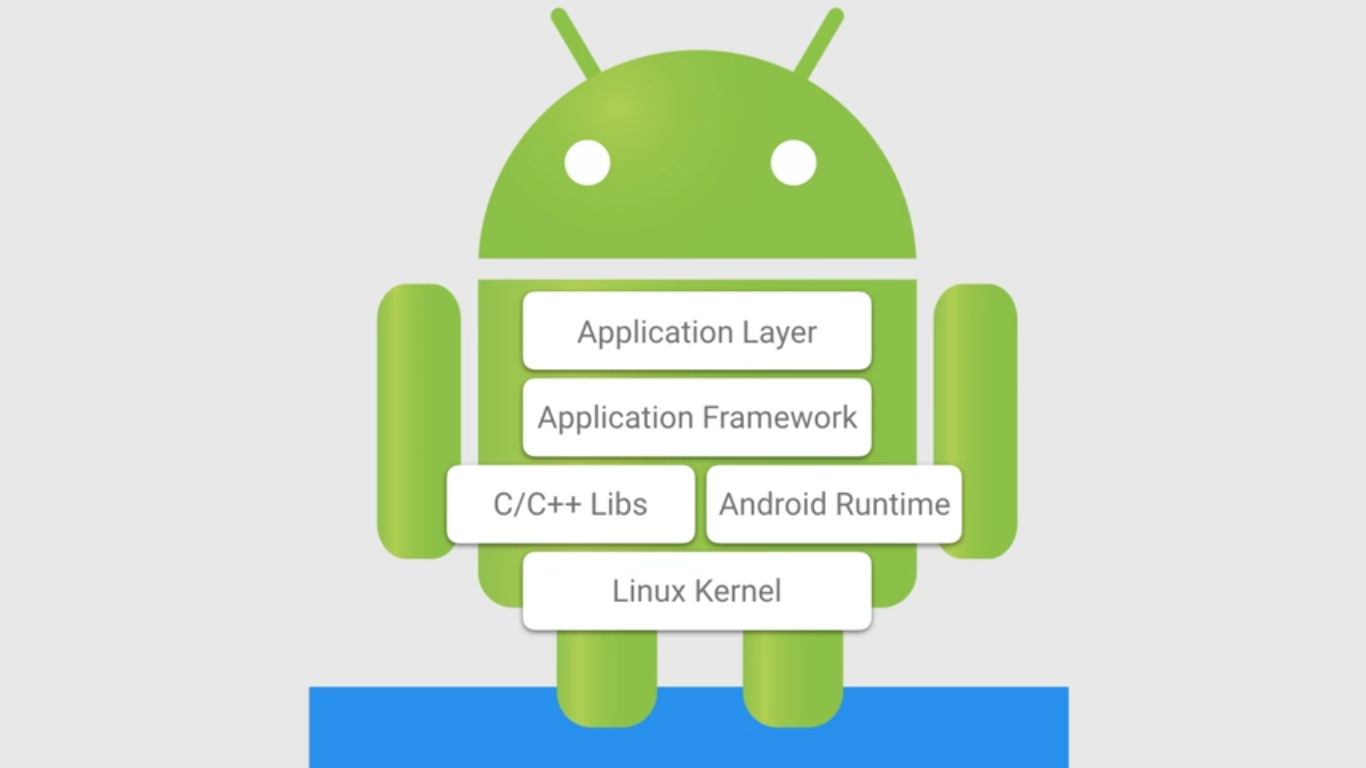
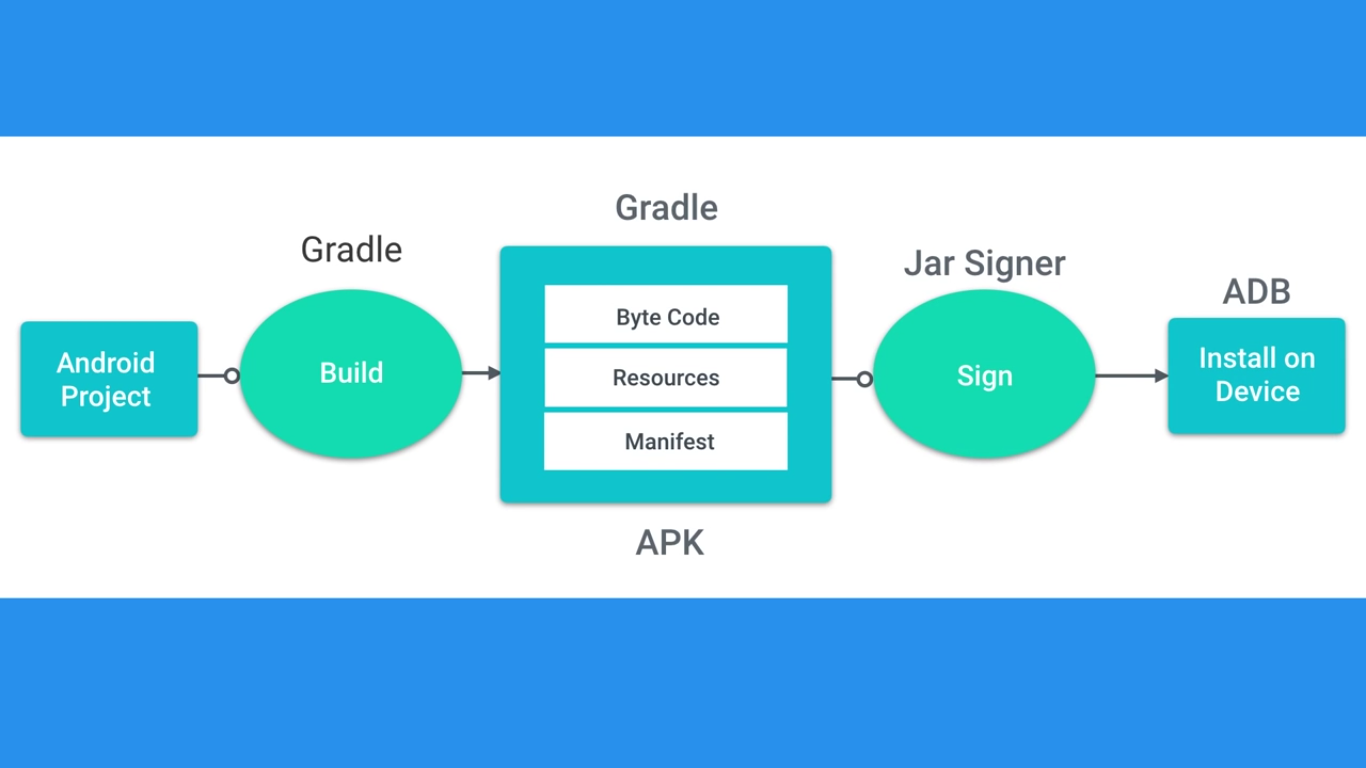
Android is a full software stack.

Gradle is the build system of choice for Android Studio.



**Android Application**

is collection of connected **components** word with each other and with **android framework**.

User perspective series of linked activity beginning from one start of launcher store in stack for user back button view pre-activity

What is activity?

Android Component responsible for most app user interaction

**Single focused** thing **user** can **do**

Responsible for creating the window that your application uses to draw and receive events from system such that touch events

Four Type of Components make our apps

Activity, Service, Broadcast Receiver, Content Provider Registered in Android Manifest

Android component is responsible for displaying the application **user interface**? It is **Activity**

Android know which **components** are **in** the application by **Android Manifest**

Types of Views UI components and Container View

**Type of View : UI Components**

There are two major categories of views. The first type are UI components that are often interactive. Here are a few examples:

| **Class Name** | **Description** |
| --- | --- |
| [TextView](https://developer.android.com/reference/android/widget/TextView.html) | Creates text on the screen; generally non interactive text. |
| [EditText](https://developer.android.com/reference/android/widget/EditText.html) | Creates a text input on the screen |
| [ImageView](https://developer.android.com/reference/android/widget/ImageView.html) | Creates an image on the screen |
| [Button](https://developer.android.com/reference/android/widget/Button.html) | Creates a button on the screen |
| [Chronometer](https://developer.android.com/reference/android/widget/Chronometer.html) | Create a simple timer on screen |

The [android.widget](https://developer.android.com/reference/android/widget/package-summary.html) package contains a list of *most* of the UI view classes available to you.

**Type of View : Container View**

The second are views called "Layout" or "Container" views. They extend from a class called [ViewGroup](https://developer.android.com/reference/android/view/ViewGroup.html). They are primarily responsible for containing a group of views and determining where they are on screen. What do I mean by "containing a group of views?". I mean that a view will be nested inside the tag of another view, like below:

A few examples of common container views are:

| **Class Name** | **Description** |
| --- | --- |
| [LinearLayout](https://developer.android.com/reference/android/widget/LinearLayout.html) | Displays views in a single column or row. |
| [RelativeLayout](https://developer.android.com/reference/android/widget/RelativeLayout.html) | Displays views positioned relative to each other and this view. |
| [FrameLayout](https://developer.android.com/reference/android/widget/FrameLayout.html) | A ViewGroup meant to contain a single child view. |
| [ScrollView](https://developer.android.com/reference/android/widget/ScrollView.html) | A FrameLayout that is designed to let the user scroll through the content in the view. |
| [ConstraintLayout](https://developer.android.com/reference/android/support/constraint/ConstraintLayout.html) | This is a newer viewgroup; it positions views in a flexible way. We’ll be exploring constraint layout later in the lesson. |

Note that layout views can be nested in one another, so you can nest a LinearLayout inside of a LinearLayout if you so choose.

**Padding and Margin**

padding and layout\_margin are two very similar attributes. Both determine the space around a View. The difference is that padding determines space within the boundaries of the view, and layout\_margin determines the space outside the boundaries of the view. For a thorough example, check out this video about [Padding and Margin](https://classroom.udacity.com/courses/ud837/lessons/4330701752/concepts/42402386170923)

**The R Class**

When your application is compiled the [R](https://developer.android.com/reference/android/R.html) class is generated. It creates constants that allow you to dynamically identify the various contents of the res folder, including layouts. To learn more, check out the documentation about [resources](https://developer.android.com/guide/topics/resources/accessing-resources.html).

ID

@+id/tv\_toy\_names

@ means that look at content android recourse

+ means that create id if it doesn’t exist

Id means that it is id not string, style or image

ConstrainLayout

<https://codelabs.developers.google.com/codelabs/constraint-layout/#0>

FrameLayout is good for simple Layout if you have one child view like listView fill entire content area

RelativeLayout if multiple items and label overlaps position other view relative to itself

LinearLayout if stack of view vertically or horizontal

ConstraintLayout if separate view relative to each other or parent view

**setText Vs append**

Another method you can use is [setText](https://developer.android.com/reference/android/widget/TextView.html#setText(int)). The difference between setText and append is that setText overwrites what was in the TextView, while append simply adds text onto whatever text was already there.

Logging display levels

**Release**

**Error obvious error**

**Warn don’t make error or crash but remain concern like low disk space for media**

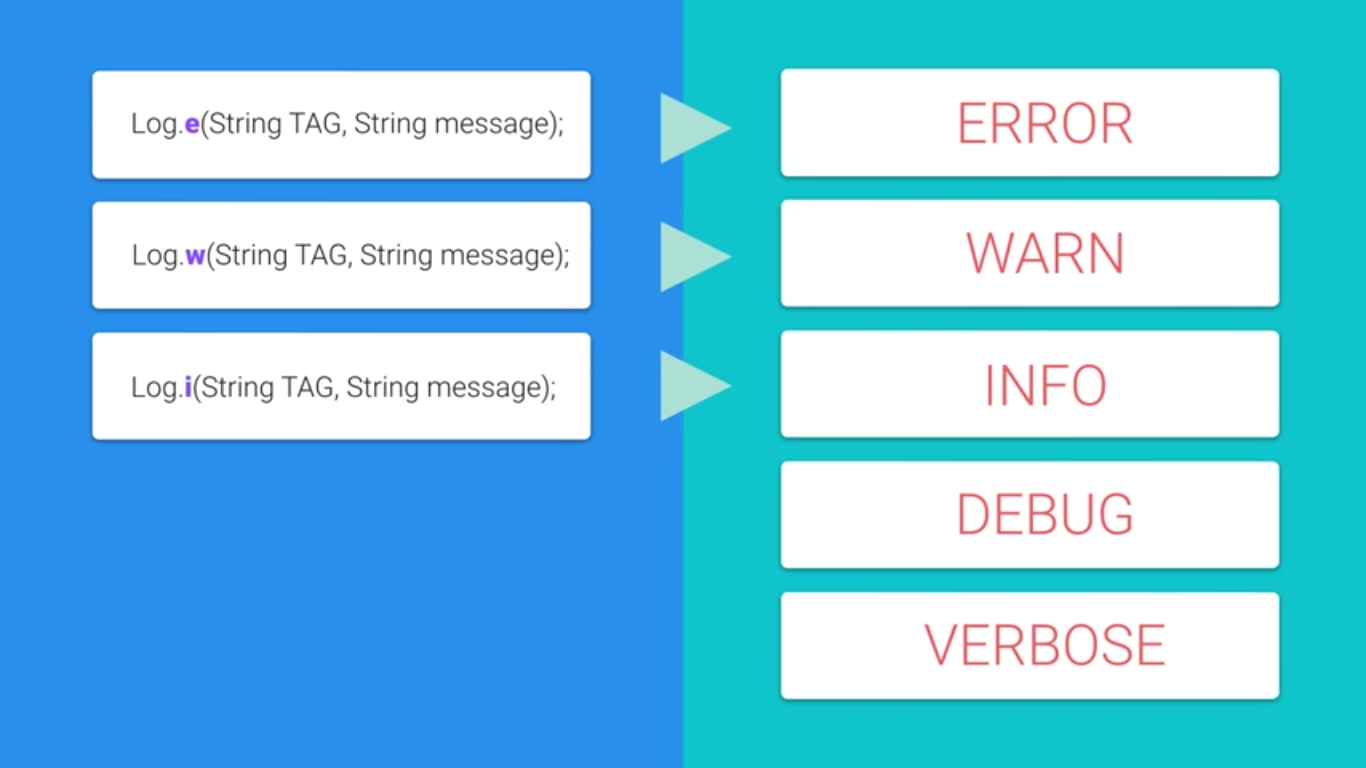
**Info pure information messages like been connected to internet**

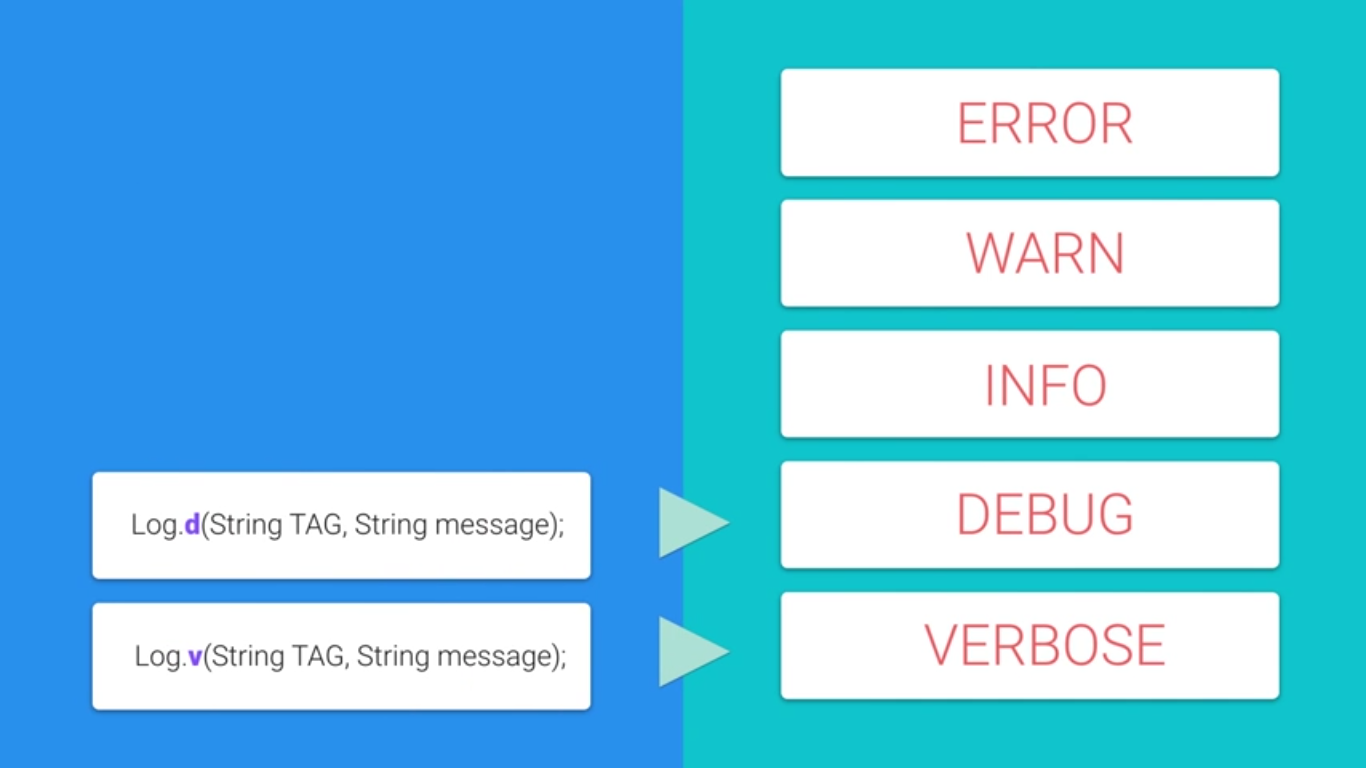
Development

Debug output URL function is composed, response for web serves

Verbose very granular information

WTF What a Terrible Failure error should never ever happened, and developer should never use it force device to output debug report





Has good links for android on GitHub

<https://api.github.com/search/repositories?q=android&sort=stars>

xliff

For example, let's look at Sunshine's **strings.xml** file:

<string name="today">Today</string>

<!-- For labelling tomorrow's forecast [CHAR LIMIT=15] -->

<string name="tomorrow">Tomorrow</string>

<!-- Date format [CHAR LIMIT=NONE] -->

<string name="format\_full\_friendly\_date">

<xliff:g id="month">%1$s</xliff:g>, <xliff:g id="day">%2$s</xliff:g>

</string>

The id of the String with the value "Today" is today and the id of the String with the value <xliff:g id="month">%1$s</xliff:g>, <xliff:g id="day">%2$s</xliff:g> is format\_full\_friendly\_date

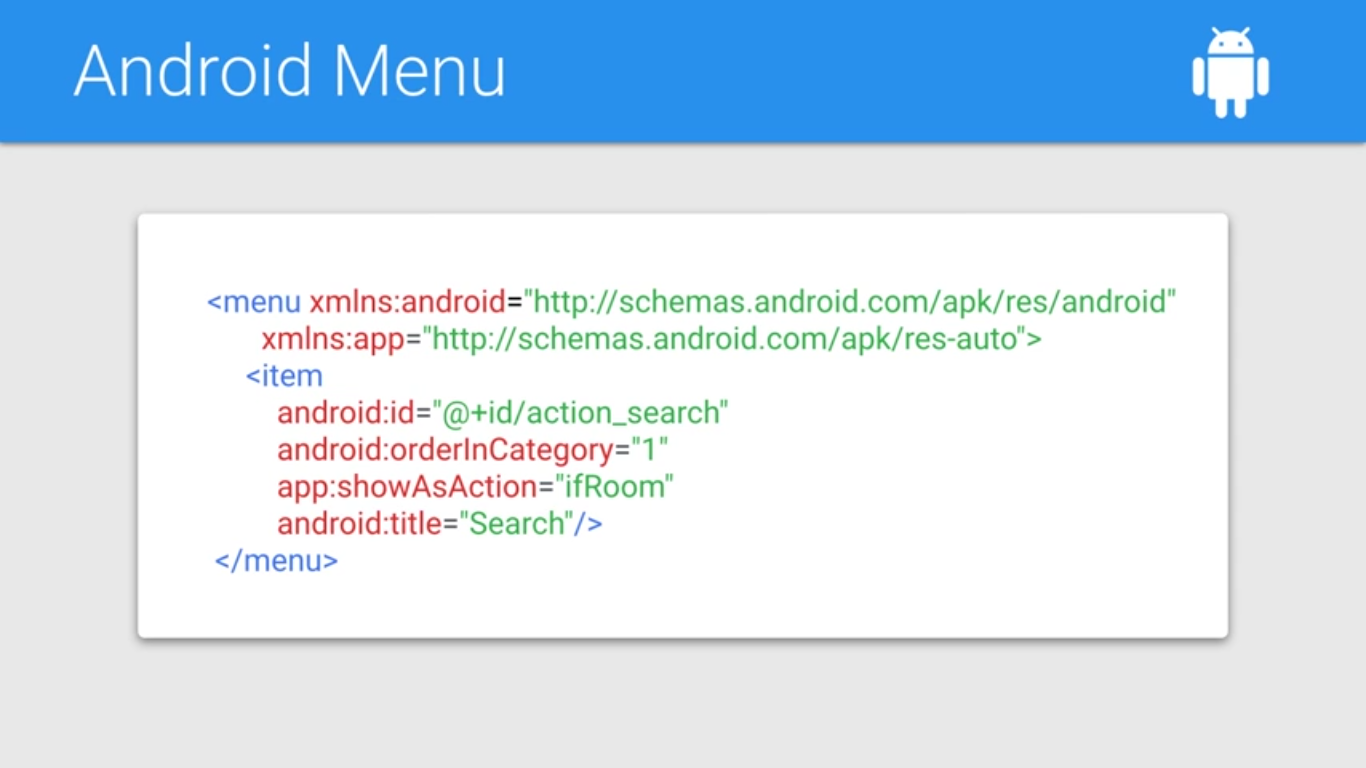
If you wanted to reference the **Today** string, you would reference it in Java by doing something like this:

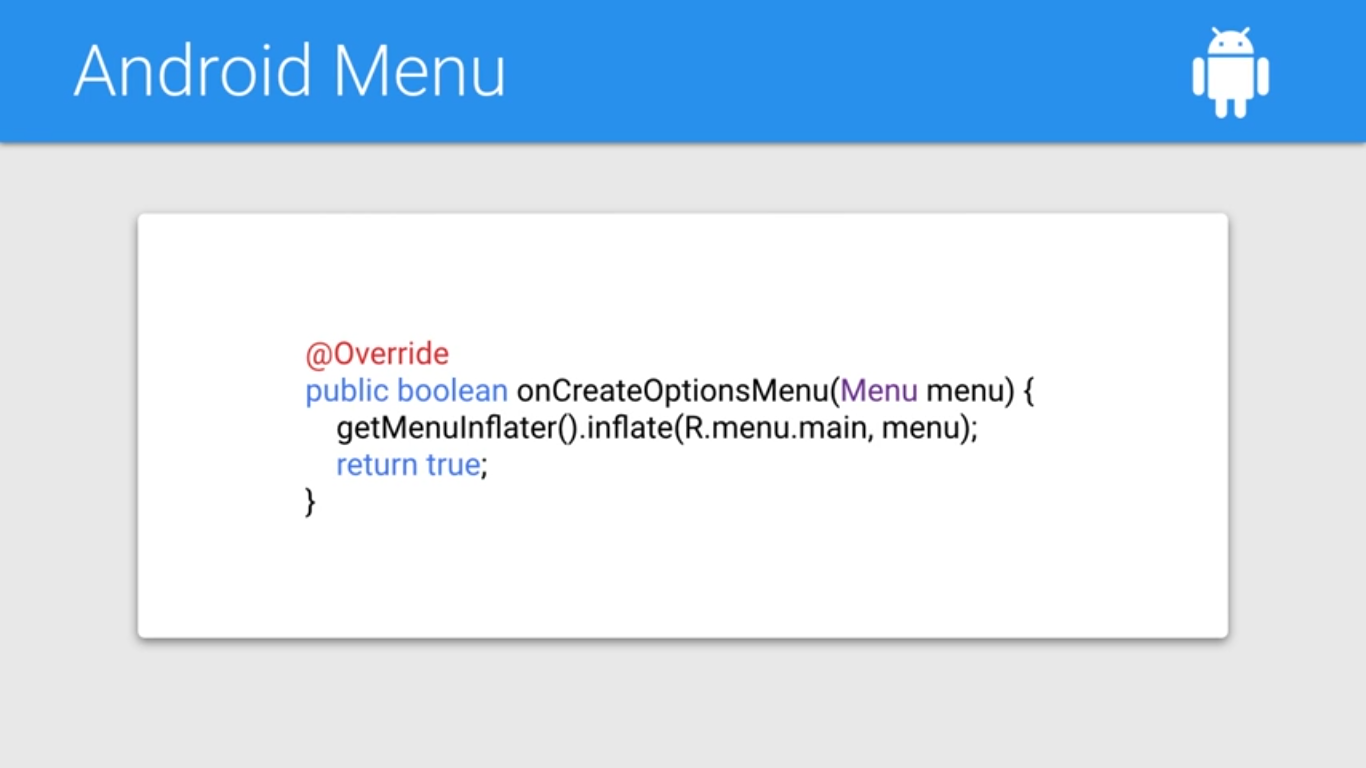
String myString = getString(R.string.today);

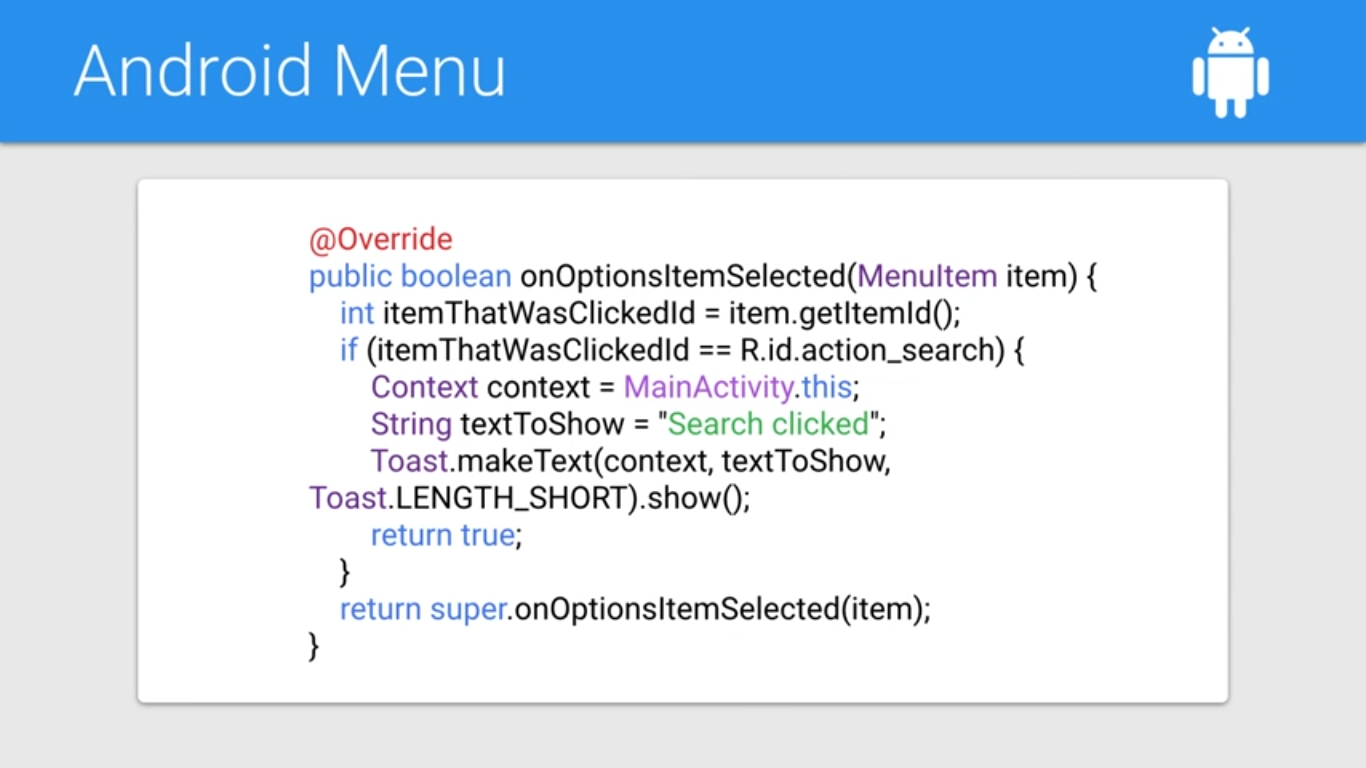
In XML, you can access a String by using the @string accessor method. For the same String defined above, you could access it like this:

<TextView text=”@string/today” />

Android Menu







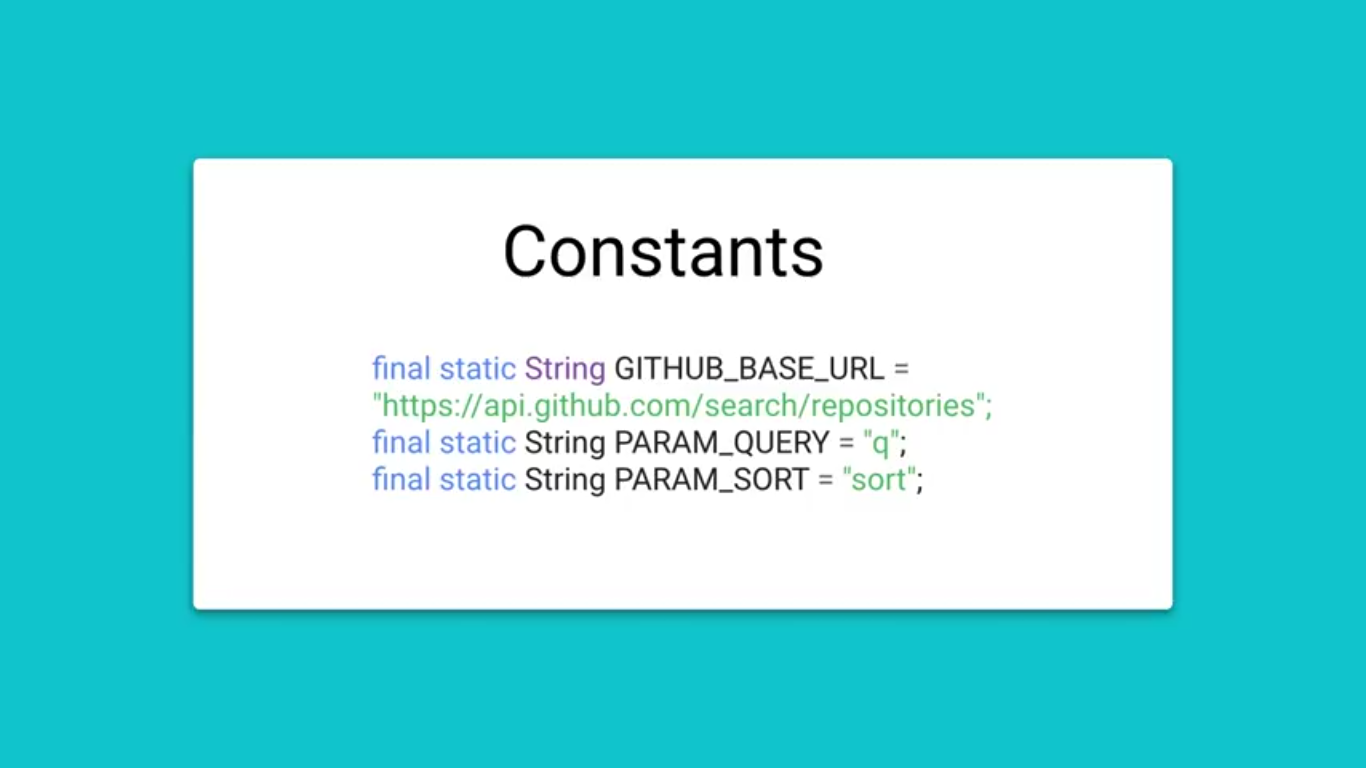
@Override  
public boolean onCreateOptionsMenu(Menu menu) {  
 getMenuInflater().inflate(R.menu.*main*, menu);  
 return true;  
}

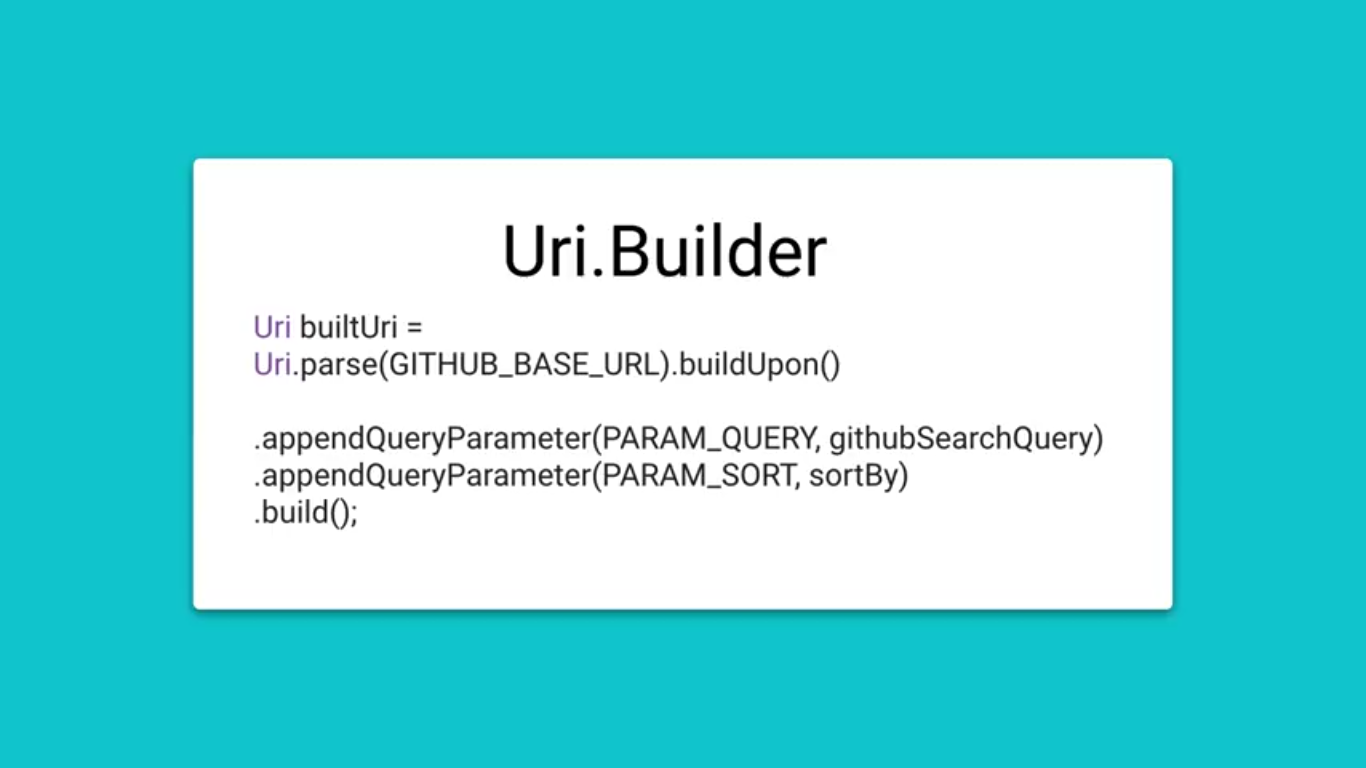
@Override  
 public boolean onOptionsItemSelected(MenuItem item) {  
 int itemThatWasClickedId = item.getItemId();  
 if (itemThatWasClickedId == R.id.*action\_search*) {  
Context context = MainActivity.this;  
 String textToShow = "Search clicked";  
 Toast.*makeText*(context, textToShow, Toast.*LENGTH\_SHORT*).show();  
 return true;  
 }  
 return super.onOptionsItemSelected(item);  
 }  
}

app:showAsAction, Vs android:showAsAction

Build URL

URL is specific class of URI







public class NetworkUtils {  
  
 final static String *GITHUB\_BASE\_URL* =  
 "https://api.github.com/search/repositories";  
  
 final static String *PARAM\_QUERY* = "q";  
  
 /\*  
 \* The sort field. One of stars, forks, or updated.  
 \* Default: results are sorted by best match if no field is specified.  
 \*/  
 final static String *PARAM\_SORT* = "sort";  
 final static String *sortBy* = "stars";  
  
 */\*\*  
 \* Builds the URL used to query Github.  
 \*  
 \** ***@param*** *githubSearchQuery The keyword that will be queried for.  
 \** ***@return*** *The URL to use to query the weather server.  
 \*/* public static URL buildUrl(String githubSearchQuery) {  
 // *TODO (1) Fill in this method to build the proper Github query URL* Uri buildUri =Uri.*parse*(*GITHUB\_BASE\_URL*).buildUpon()  
 .appendQueryParameter(*PARAM\_QUERY*,githubSearchQuery)  
 .appendQueryParameter(*PARAM\_SORT*,*sortBy*).build();  
  
 URL url = null;  
 try {  
 url = new URL(buildUri.toString());  
 }catch (MalformedURLException e)  
 {  
 e.printStackTrace();  
 }  
 return url;  
 }  
}

Naming id in XML they use

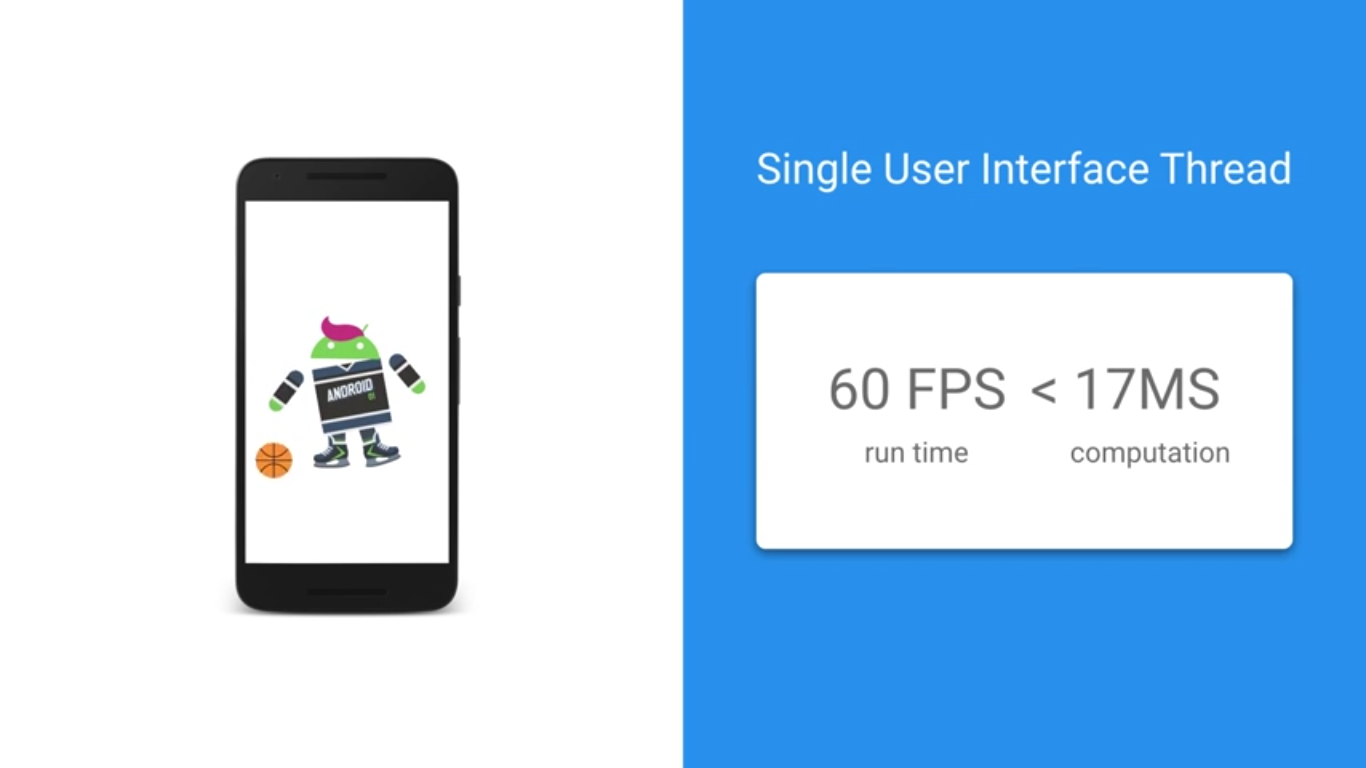
tv\_ for TextView

et\_ for EditText

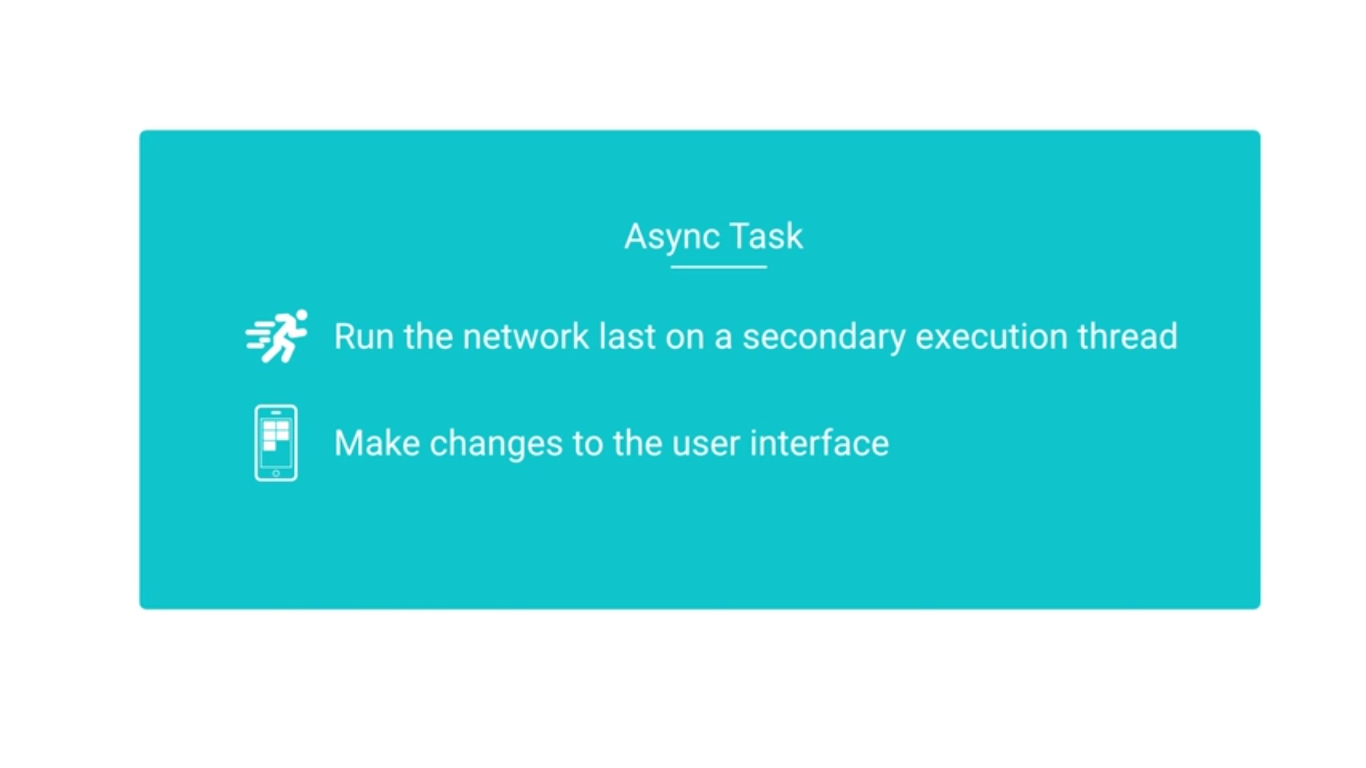
convert input stream to string <https://stackoverflow.com/questions/309424/read-convert-an-inputstream-to-a-string>

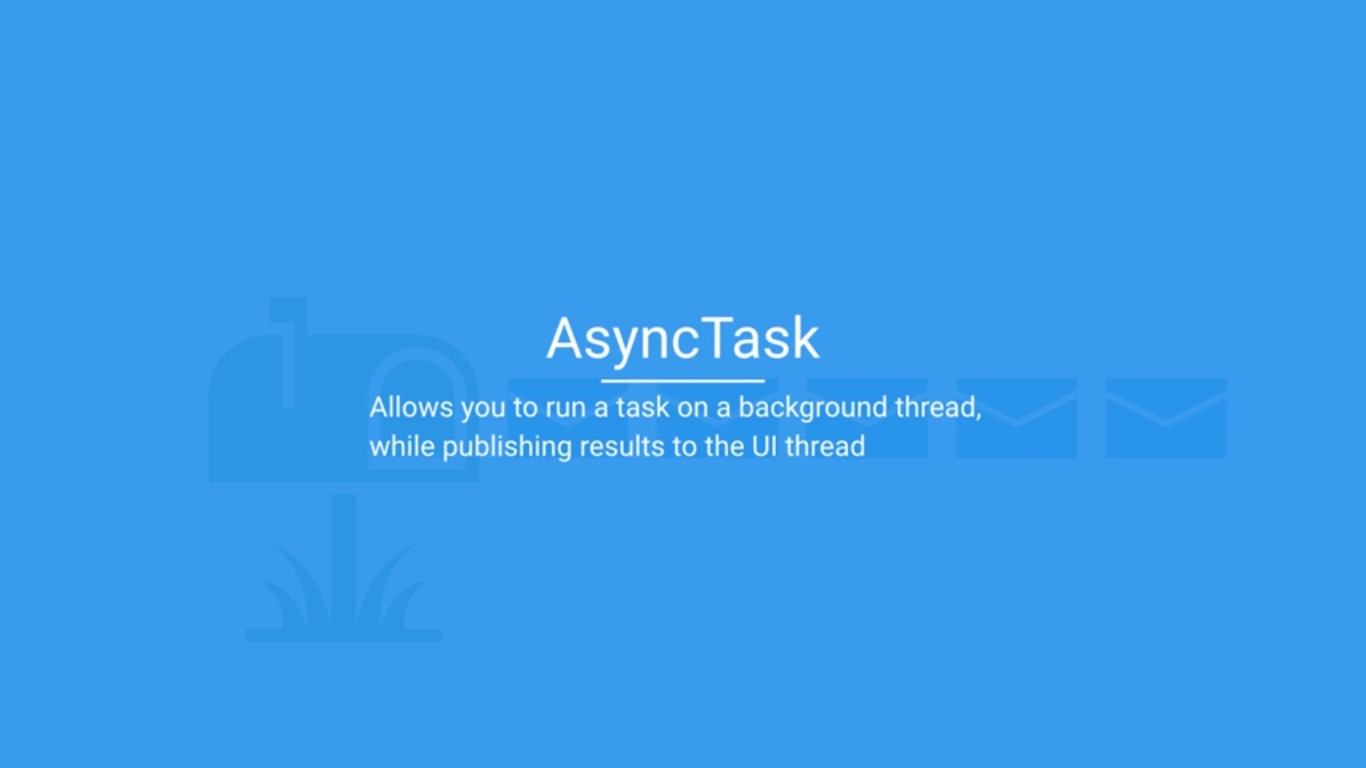
Android throws an exception then try to access the network on the main thread

Android doing multiple things at once multitask

Each android app can be divided into multiple threads of execution that can all run concurrently on modern android devices, these threads of execution can be scheduled by the OS to run on different CPU cores but system might choose to time slice the amount of single CPU which means running each for a period of time either way to you application it appears as if the threads are running concurrently to make things easier on the developers **android apps have a single user interface thread** this thread is responsible for getting events from the various sensors and setting up the next frame to draw to run **ideal 60 frames per second** we need to make sure that all computation between draws take less than **17 milliseconds** which is pretty fast in other words we want to do as little as possible on this main thread but networking can take seconds which means your app would be frozen for all that time if the networking call is on the **main thread the user couldn’t interact with it after five seconds of ignoring user input android would actually prompt the user to close your app to user network** we need to run the **network task on a secondary execution thread** but we need to **make changes to use interface** like setting the text view to show the result in the **UI thread** fortunately android provides a helpful framework pattern to do this **Async task**

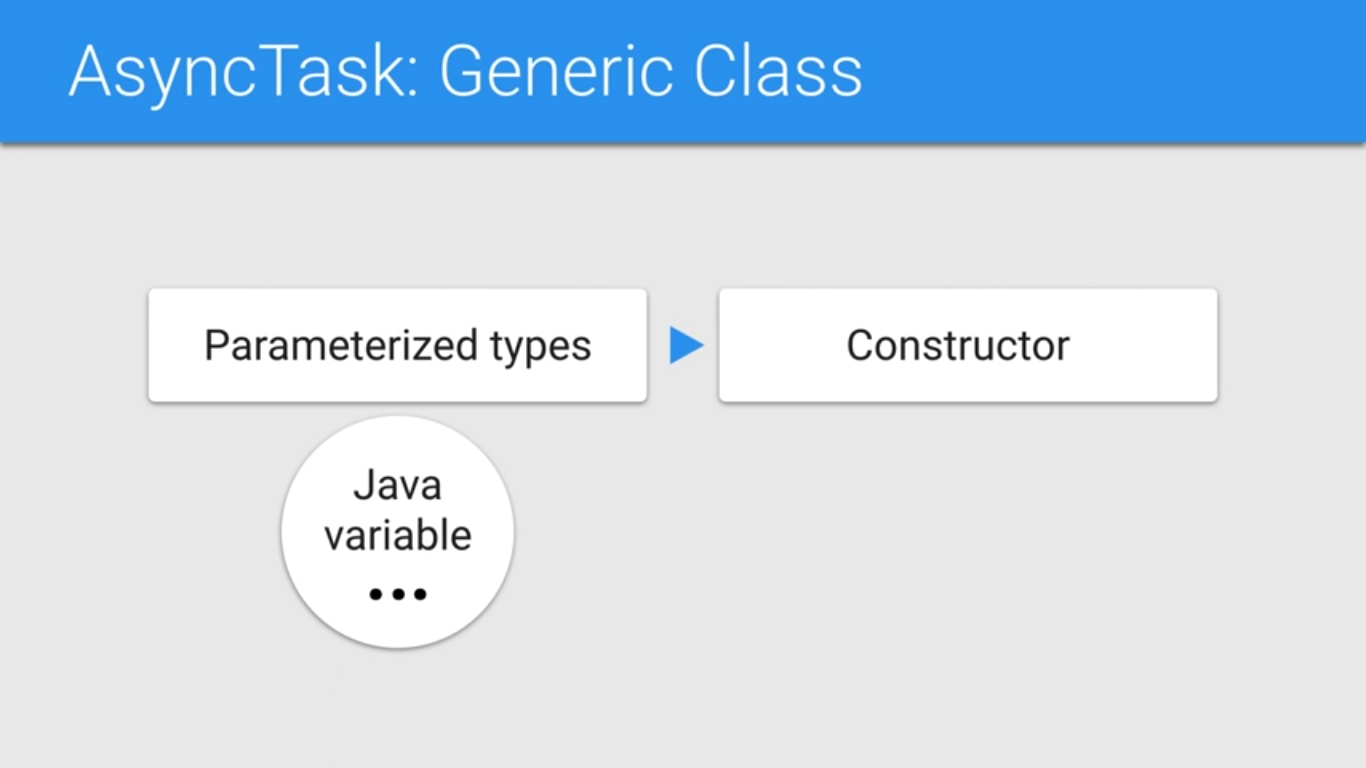
AsyncTask





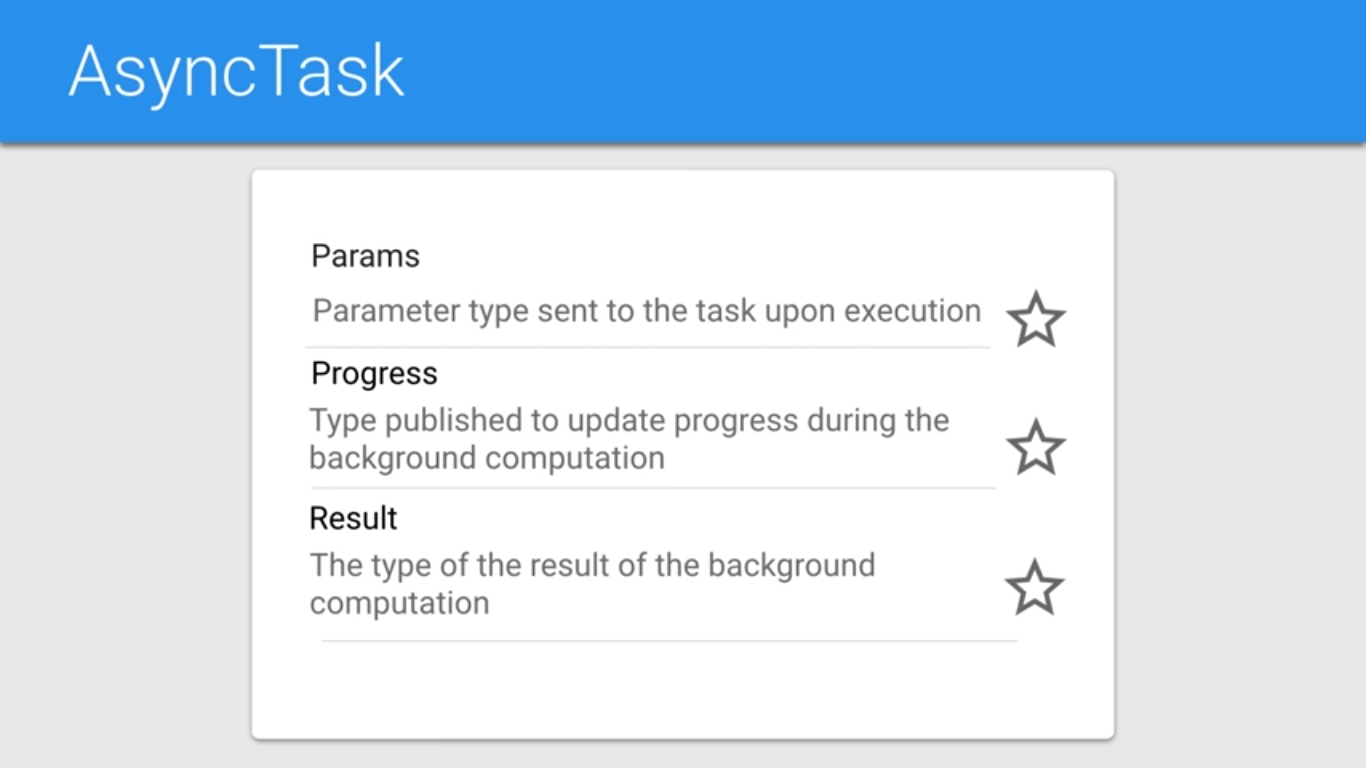
UI thread

UI thread has a message queue and handler that allows you to send a process runnable objects and message often from other threads AsyncTask wraps this behavior with a reasonably straightforward interface

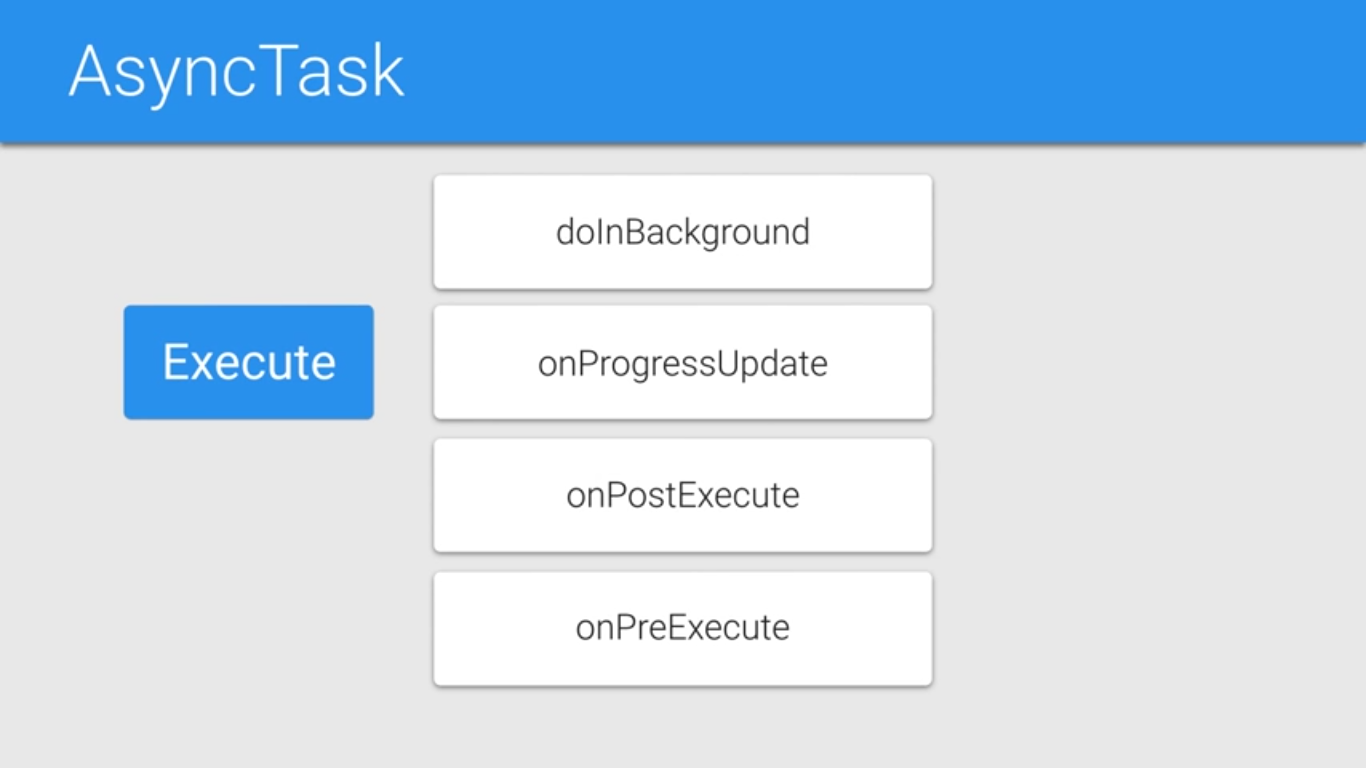
AsyncTask is a generic class

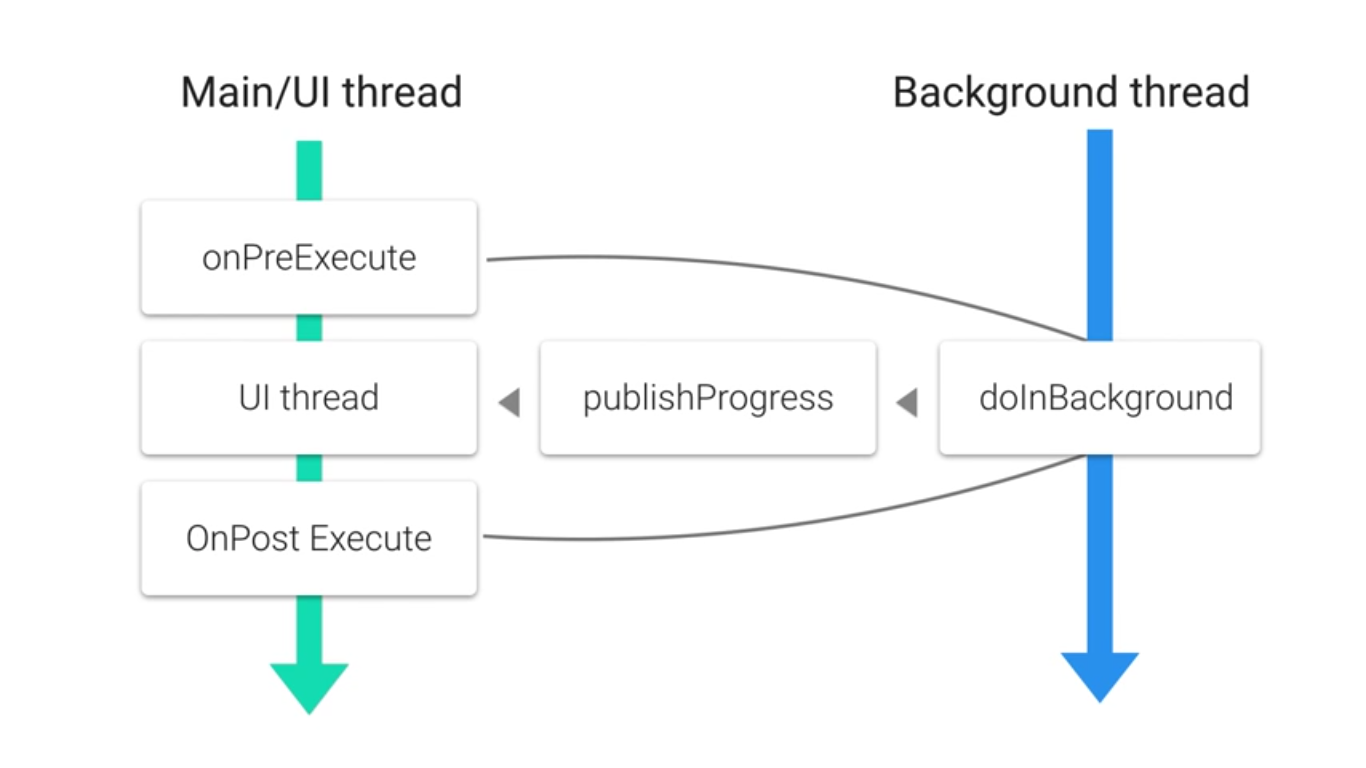
Meaning that it takes parameterized types in its constructors

Each one of these generic parameters is to find as a java variable argument with three dots which means that it is technically passed as an array in java word

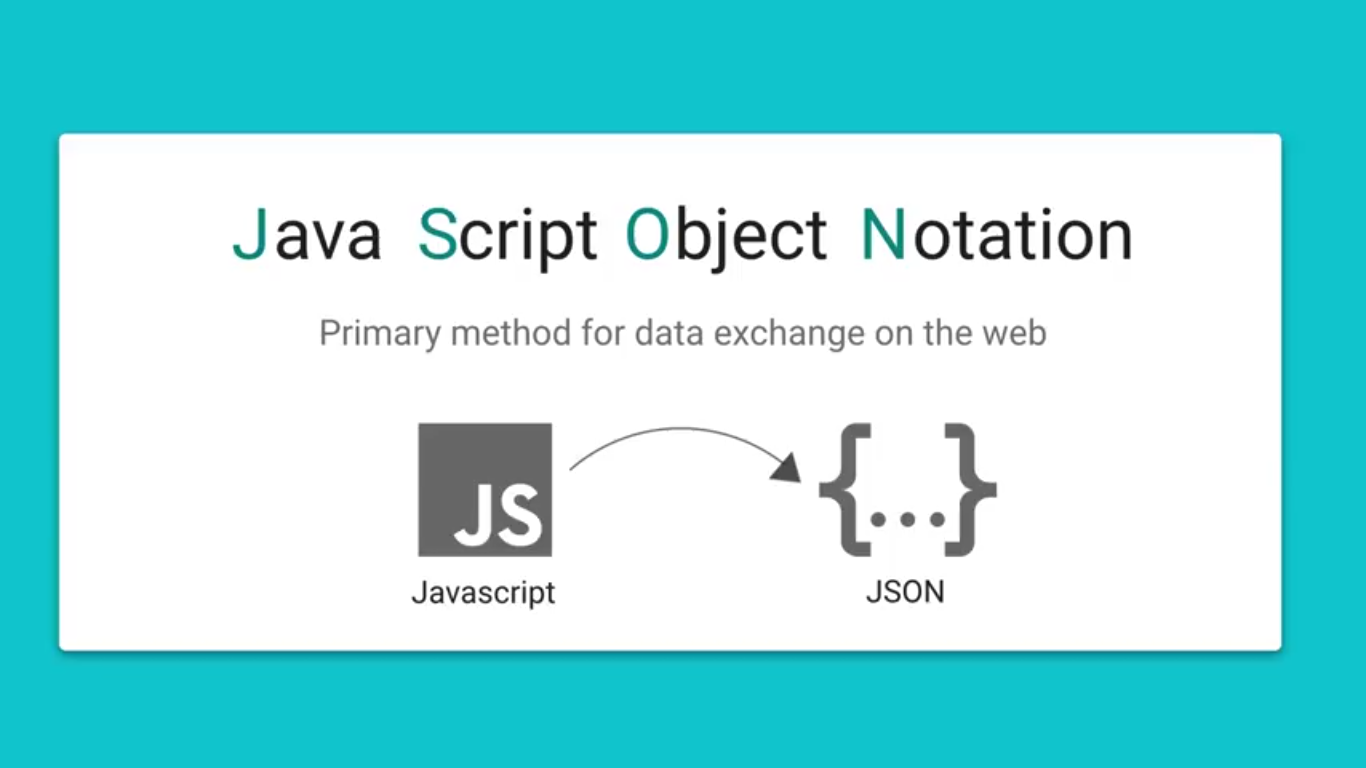


These three parameters correspond to three primary function you can override in Async Task





JSON



**JSON** **J**ava **S**cript **O**bject **N**otation

Primary method for data exchange on web

Because it’s format is syntactically identical to code for creating JavaScript Objects

This means JavaScript programs can use standard JavaScript functions to read JSON data

Data

temp

min = 11.34

max = 19.01

weather

id = 801

condition = Clouds

description = "few clouds"

pressure = 1023.51

humidity = 87

JSON Object

{

"temp": {

"min":"11.34",

"max":"19.01"

},

"weather": {

"id":"801",

"condition":"Clouds",

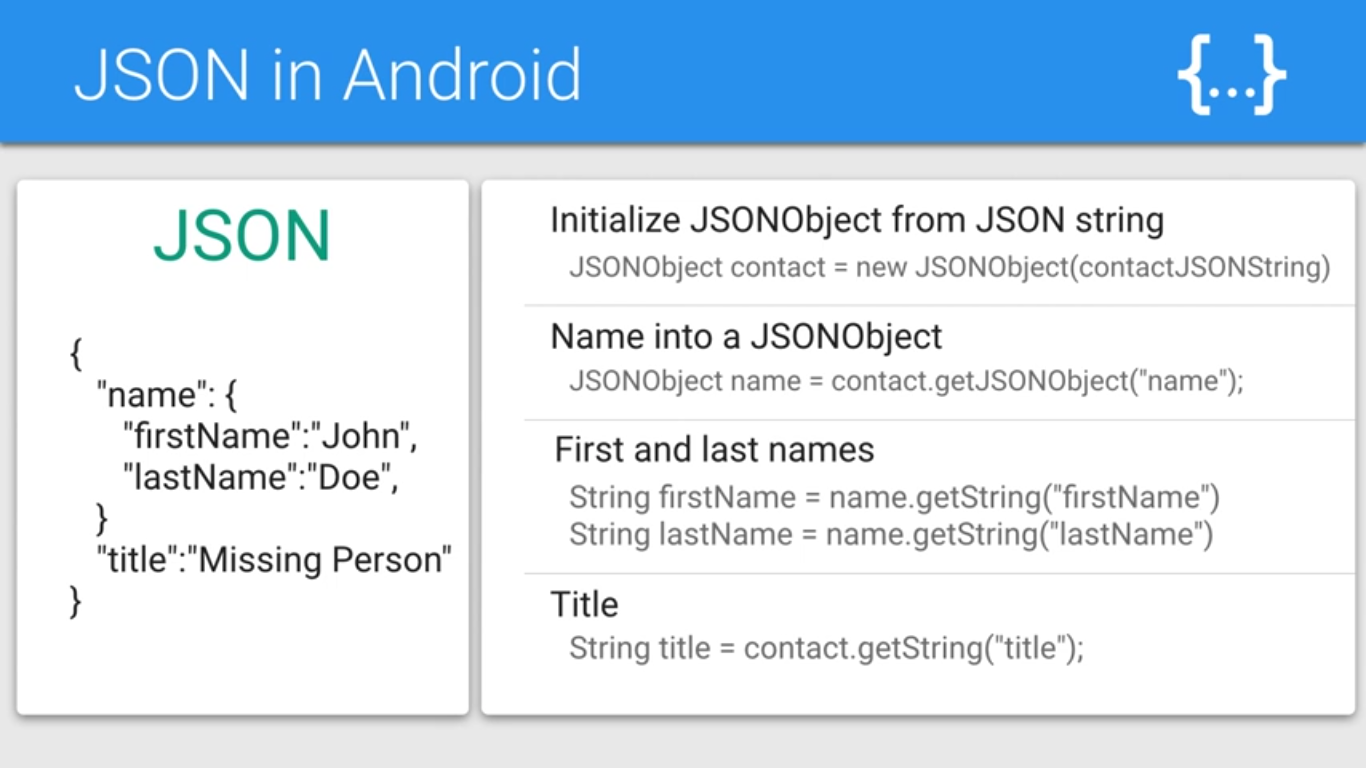
"description":"few clouds"

},

"pressure":"1023.51",

"humidity":"87"

}



Note: the comma after "Doe" should appear after the closing braces.

JSONObject contact = new JSONObject(contactJSONString)

JSONObject name = contact.getJSONObject(“name”);

String firstName = name.getString(“firstName”);

String lastName = name.getString(“lastName”);

String title = contact.getString(“title”);

Given the following JSON, write a function to retrieve the weather "condition".

{

"temp": {

"min":"11.34",

"max":"19.01"

},

"weather": {

"id":"801",

"condition":"Clouds",

"description":"few clouds"

},

"pressure":"1023.51",

"humidity":"87"

}

answer:

String getCondition(String JSONString) {

JSONObject forecast = new JSONObject(JSONString);

JSONObject weather = forecast.getJSONObject("weather");

return weather.getString("condition");

}

Short cuts

CTR + O -> methods that you want to override

Android Comment -> Ctrl + /

Arrange XML -> Ctrl + Alt + L

Prediction -> Ctrl + Space

Ctrl + O -> show the available methods to override

ALT + Insert -> automatically “Generate a method (Getters, Setters, Constructors, toString, etc..)”. On Windows

The difference between [ and {– (Square brackets and Curly brackets)

In general, all the JSON nodes will start with a square bracket or with a curly bracket. The difference between [ and { is, the square bracket ([) represents starting of an **JSONArray** node whereas curly bracket ({) represents **JSONObject**. So while accessing these nodes we need to call appropriate method to access the data.

If your JSON node starts with *[*, then we should use *getJSONArray()* method. Same as if the node starts with *{*, then we should use *getJSONObject()* method.

**Recycler View**

Every part of Recycler view is completely modeler

Recycler view has an **adapter** that is used to **provider the recycler view with new views when needed**

This **Adapter** is also used to **bind data from some Data Source to the views**

**The adapter sends the views to a RecyclerView in an object called ViewHolder**

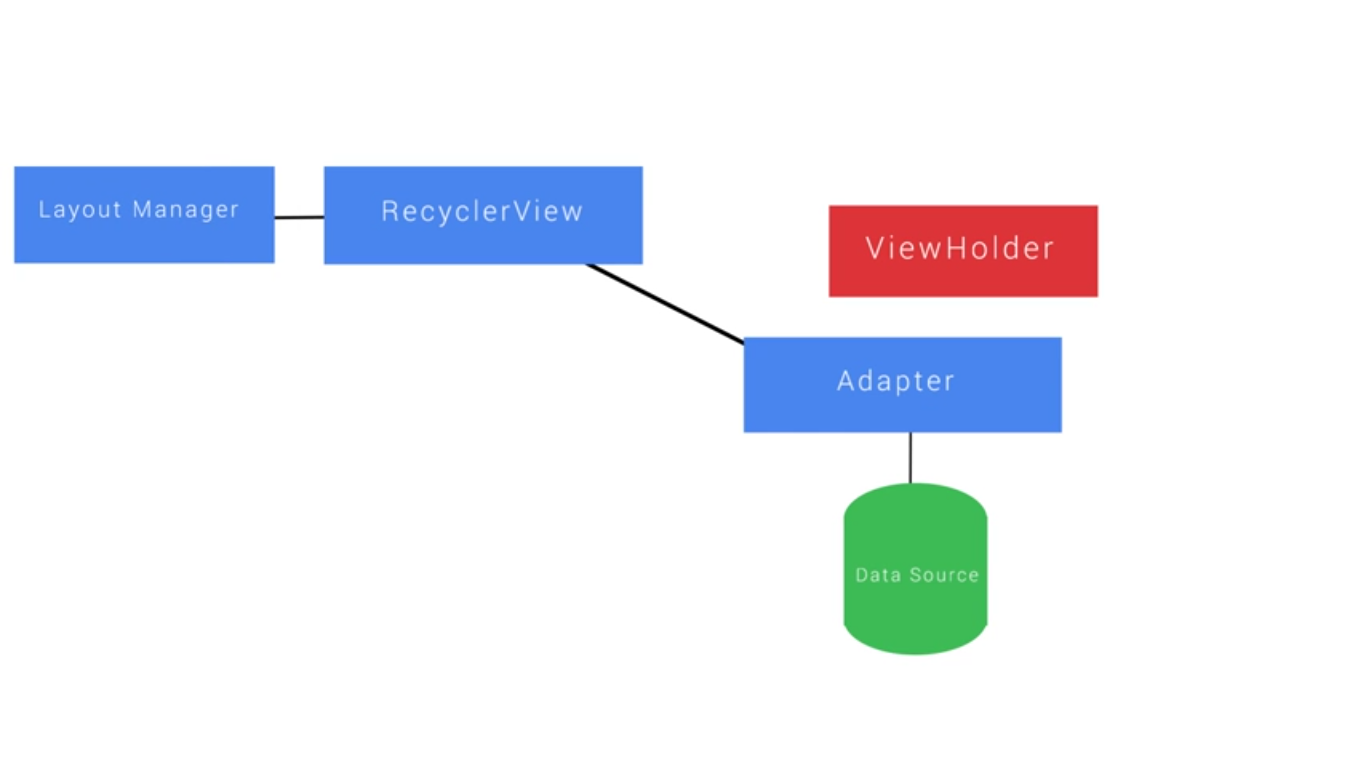
**The ViewHolder contains a reference to the root view object from the item and your are expected to user it to cache the view objects represented in the layout to make it less costly to update the views with new data**

This way find a view by id getscalled only for **each data view** whenever that new item is created **and not each time you want to populate the views in the item with data**

**The layout manager then tells the RecyclerView how to lay out all those views, it could be vertically scrolling, horizontal, staggered list or grid**

You get some awesome animations for item insertion and removal for free because it is based on data source it is easy for apps using recycler view to allow for filtering a list of items

Older was list View it is mostly become outdated thanks to RecyclerView



ViewHolder

Use to cache the view objects that you’re going to be populating with data or images

When Recycler View is first being populated you call **findViewById()** each view that will be showing data from the adaptor this search can be expensive, especially if you have a hierarchy of views in your **list item** so it’s best to do it once and cache those views in view holder this way you can access those view later without having to look them up repeatedly if you don’t use a view holder calling findViewById() for each view can get out of hand pretty quickly

**SP Vs DP**

Not actually pixel sizes but automatically scaled to be the same approximate physical size regardless of the density of the pixels on the screen

Dp when we’ve referred to dimensions in android layouts

The first andoid phones were 160 dpi and 1dp would equal 1 pixel on these devices

The conversion of dp units to screen pixels is

Px = dp \* (dpi /160)

dpi (dots per inch).

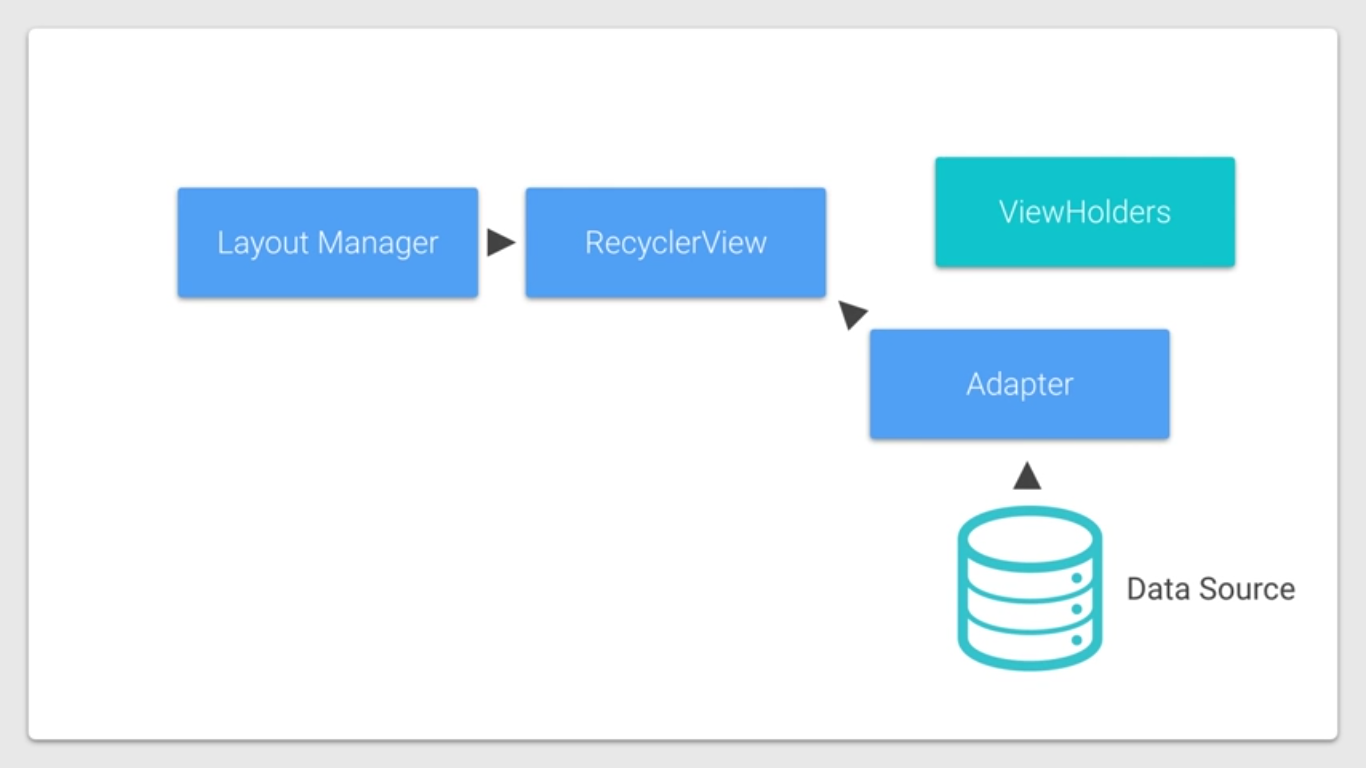
SP scaled based on user preferences Typically the text size preference if you want your text to scale up with these preferences which is important for accessibility make sure to use SP for fonts and widgets that have hard coded sizes that contain fonts

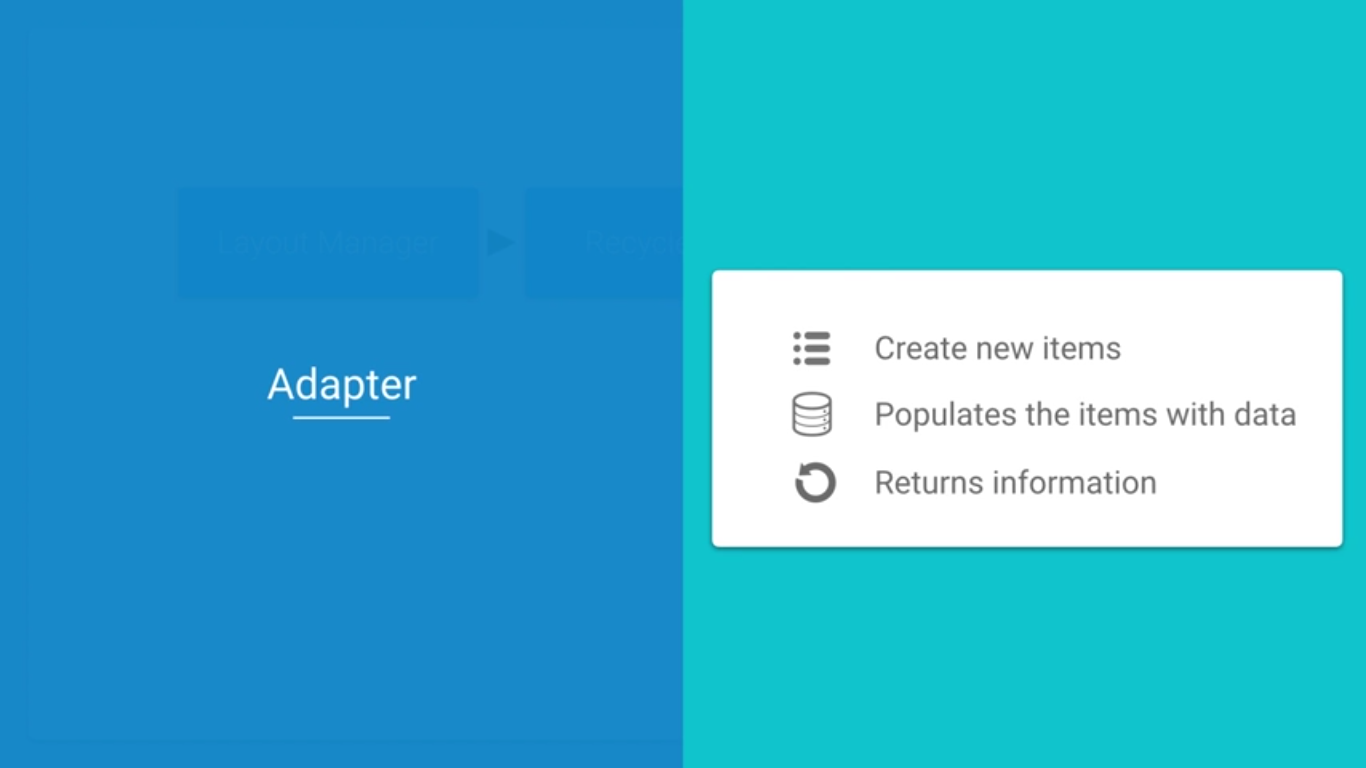
Adapter

Adapter is called by RecyclerView to create new item in form of ViewHolders

It is also populating or binds these items with data

return information about the data such as how many items there are in a given data source this data could come from an arraylist , JSON or any data source you can model





Adapter requires us to override three functions

First **onCreateViewHolder**

Which is called when the RecyclerView instantiates a new ViewHolder instance

Second **onBindViewHolder**

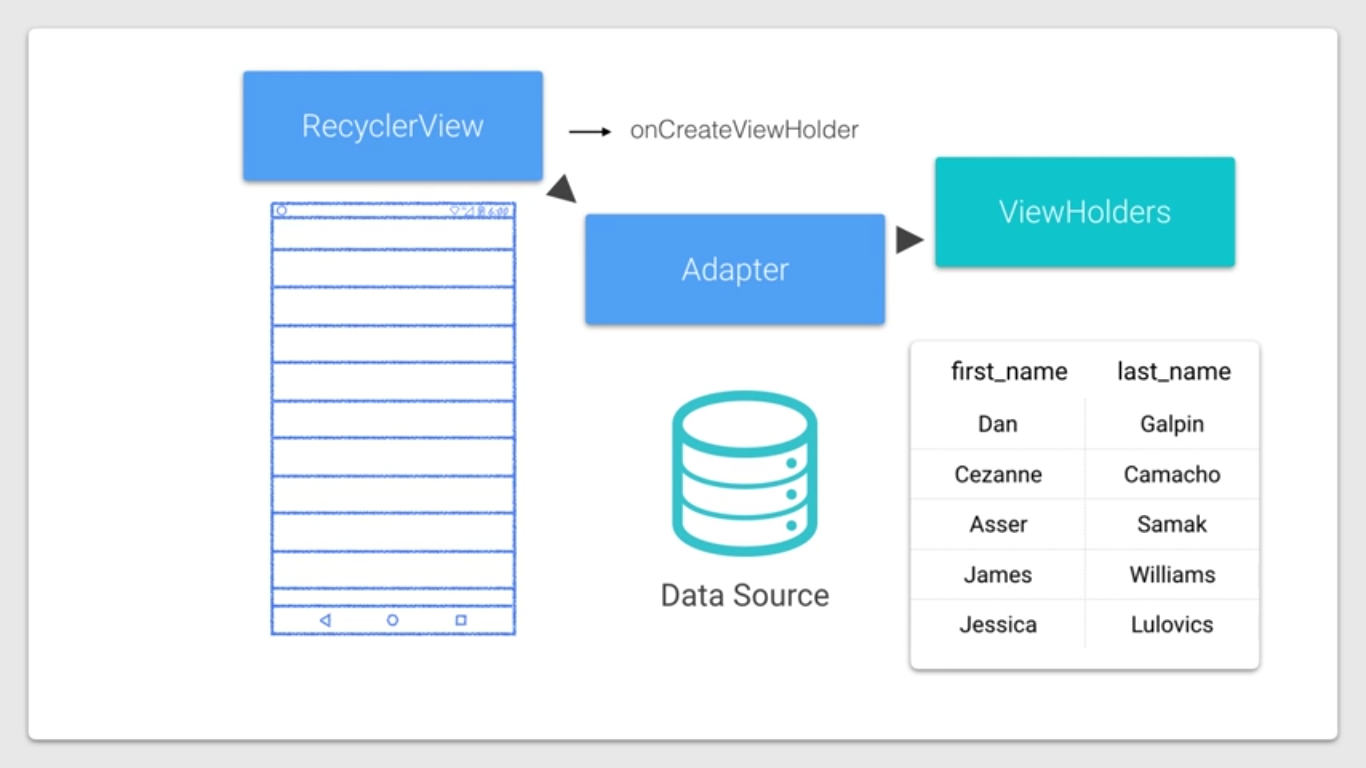
Which is called when RecyclerView wants to populate the view with data from our model so that user can see it effectively binding it to the data source

Third **getItemCount**

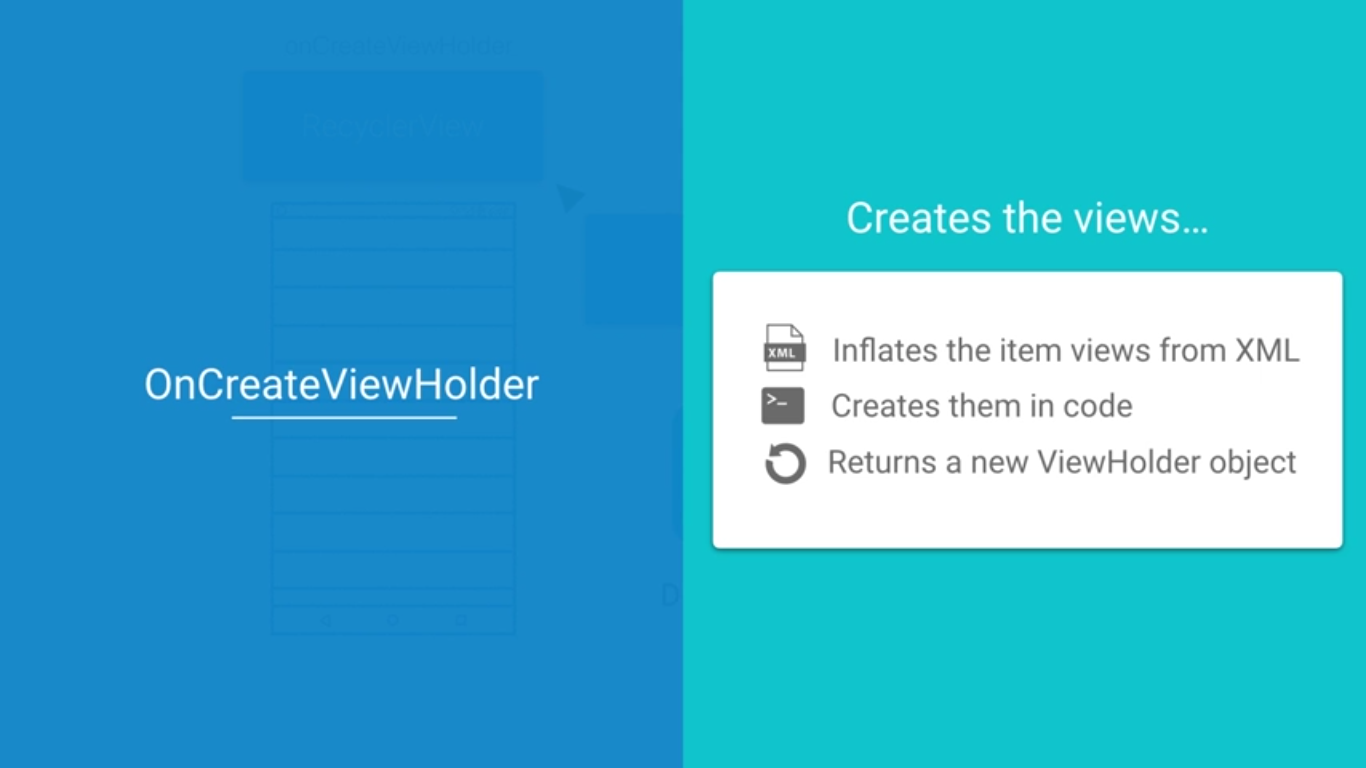
Which returns number of items in our data source

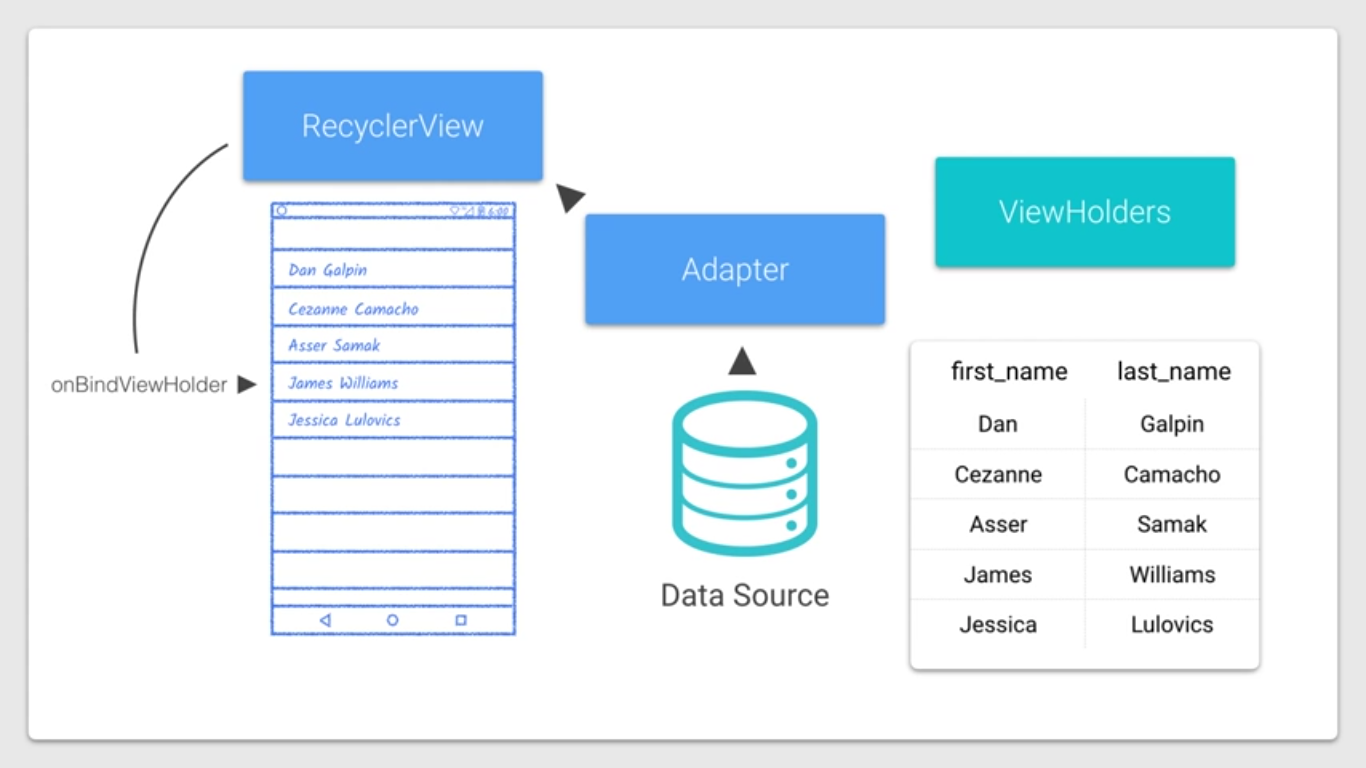
RecyclerView and data source

When the RecyclerView is being laid out and draw it will first ask the adapter number of items that it will be displaying it may ask multiple times during the layout process so it’s important that this be a fast operation then the RecyclerView will then ask the Adapter to create ViewHolder objects and in the process inflate individual item views from their corresponding XML by calling onCreateViewHolder it will create as many ViewHolders as are necessary to either display all of the items or fill and scroll the screen which option requires fewer ViewHolder



onCreateViewHolder function is responsible for creating the views either by inflating the item views from XML or creating the in code it then returns a new ViewHolder objects associated with this new view





*/\*\*  
 \*  
 \* This gets called when each new ViewHolder is created. This happens when the RecyclerView  
 \* is laid out. Enough ViewHolders will be created to fill the screen and allow for scrolling.  
 \*  
 \** ***@param*** *parent The ViewGroup that these ViewHolders are contained within.  
 \** ***@param*** *viewType If your RecyclerView has more than one type of item (which ours doesn't) you  
 \* can use this viewType integer to provide a different layout. See  
 \* {****@link*** *android.support.v7.widget.RecyclerView.Adapter#getItemViewType(int)}  
 \* for more details.  
 \** ***@return*** *A new NumberViewHolder that holds the View for each list item  
 \*/*@Override  
public NumberViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {  
  
 Context context = parent.getContext();  
 int layoutIdForListItem = R.layout.*number\_list\_item*;  
 LayoutInflater inflater = LayoutInflater.*from*(context);  
 boolean shouldAttachToParentImmediately = false;  
  
 View view = inflater.inflate(layoutIdForListItem, parent, shouldAttachToParentImmediately);  
 NumberViewHolder viewHolder = new NumberViewHolder(view);  
  
 return viewHolder;  
}

short