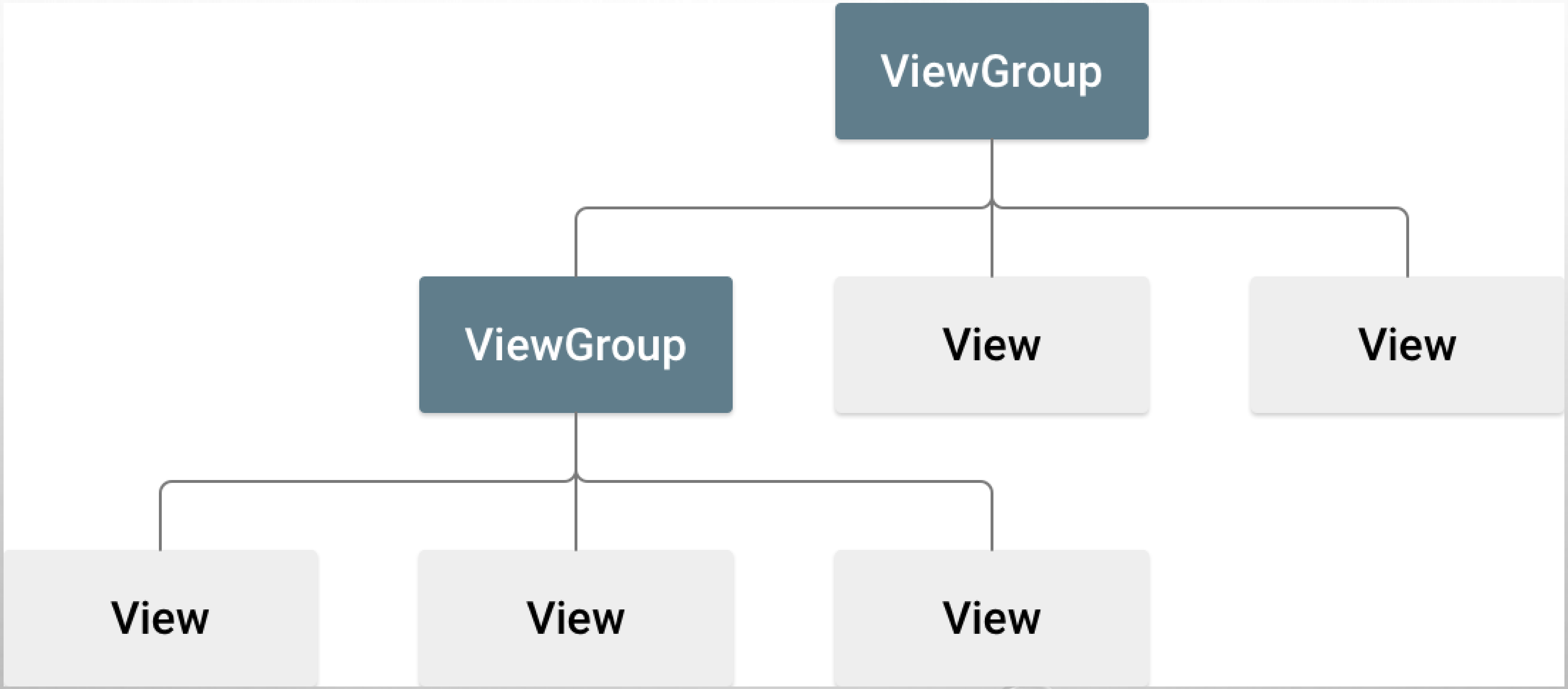
USER INTERFACE

- ALL USER INTERFACE ELEMENTS IN AN ANDROID APP ARE BUILT USING VIEW AND VIEWGROUP OBJECTS.
- A VIEW IS AN OBJECT THAT DRAWS SOMETHING ON THE SCREEN THAT THE USER CAN INTERACT WITH.
- A VIEWGROUP IS AN OBJECT THAT HOLDS OTHER VIEW (AND VIEWGROUP) OBJECTS IN ORDER TO DEFINE THE LAYOUT OF THE INTERFACE.
- TO DECLARE YOUR LAYOUT, YOU CAN INSTANTIATE VIEW OBJECTS IN CODE AND START BUILDING A TREE, BUT THE EASIEST AND MOST EFFECTIVE WAY TO DEFINE YOUR LAYOUT IS WITH AN XML FILE.
- XML OFFERS A HUMAN-READABLE STRUCTURE FOR THE LAYOUT, SIMILAR TO HTML.

CONT …



LAYOUTS

- A LAYOUT DEFINES THE VISUAL STRUCTURE FOR A USER INTERFACE, SUCH AS THE UI FOR AN ACTIVITY OR APP WIDGET.
- YOU CAN DECLARE A LAYOUT IN TWO WAYS:
  - **DECLARE UI ELEMENTS IN XML.**
  - **INSTANTIATE LAYOUT ELEMENTS AT RUNTIME**
- THE ADVANTAGE TO DECLARING YOUR UI IN XML IS THAT IT ENABLES YOU TO BETTER SEPARATE THE PRESENTATION OF YOUR APPLICATION FROM THE CODE THAT CONTROLS ITS BEHAVIOR.
- YOUR UI DESCRIPTIONS ARE EXTERNAL TO YOUR APPLICATION CODE, WHICH MEANS THAT YOU CAN MODIFY OR ADAPT IT WITHOUT HAVING TO MODIFY YOUR SOURCE CODE AND RECOMPILE.

ATTRIBUTES

- EVERY VIEW AND VIEWGROUP OBJECT SUPPORTS THEIR OWN VARIETY OF XML ATTRIBUTES.
- SOME ATTRIBUTES ARE SPECIFIC TO A VIEW OBJECT (FOR EXAMPLE, TEXTVIEW SUPPORTS THE TEXTSIZE ATTRIBUTE), BUT THESE ATTRIBUTES ARE ALSO INHERITED BY ANY VIEW OBJECTS THAT MAY EXTEND THIS CLASS.
- SOME ARE COMMON TO ALL VIEW OBJECTS, BECAUSE THEY ARE INHERITED FROM THE ROOT VIEW CLASS (LIKE THE ID ATTRIBUTE).
- AND, OTHER ATTRIBUTES ARE CONSIDERED "LAYOUT PARAMETERS," WHICH ARE ATTRIBUTES THAT DESCRIBE CERTAIN LAYOUT ORIENTATIONS OF THE VIEW OBJECT, AS DEFINED BY THAT OBJECT'S PARENT VIEWGROUP OBJECT.

CONT …

- ID
  - ANY VIEW OBJECT MAY HAVE AN INTEGER ID ASSOCIATED WITH IT, TO UNIQUELY IDENTIFY THE VIEW WITHIN THE TREE.
  - WHEN THE APPLICATION IS COMPILED, THIS ID IS REFERENCED AS AN INTEGER, BUT THE ID IS TYPICALLY ASSIGNED IN THE LAYOUT XML FILE AS A STRING, IN THE ID ATTRIBUTE.
  - THIS IS AN XML ATTRIBUTE COMMON TO ALL VIEW OBJECTS (DEFINED BY THE VIEW CLASS).
  - THE SYNTAX FOR AN ID, INSIDE AN XML TAG IS:
    - **ANDROID:ID="@+ID/MY_BUTTON"**
  - THE AT-SYMBOL (@) AT THE BEGINNING OF THE STRING INDICATES THAT THE XML PARSER SHOULD PARSE AND EXPAND THE REST OF THE ID STRING AND IDENTIFY IT AS AN ID RESOURCE.
  - THE PLUS-SYMBOL (+) MEANS THAT THIS IS A NEW RESOURCE NAME THAT MUST BE CREATED AND ADDED TO OUR RESOURCES (IN THE R.JAVA FILE).

CONT …

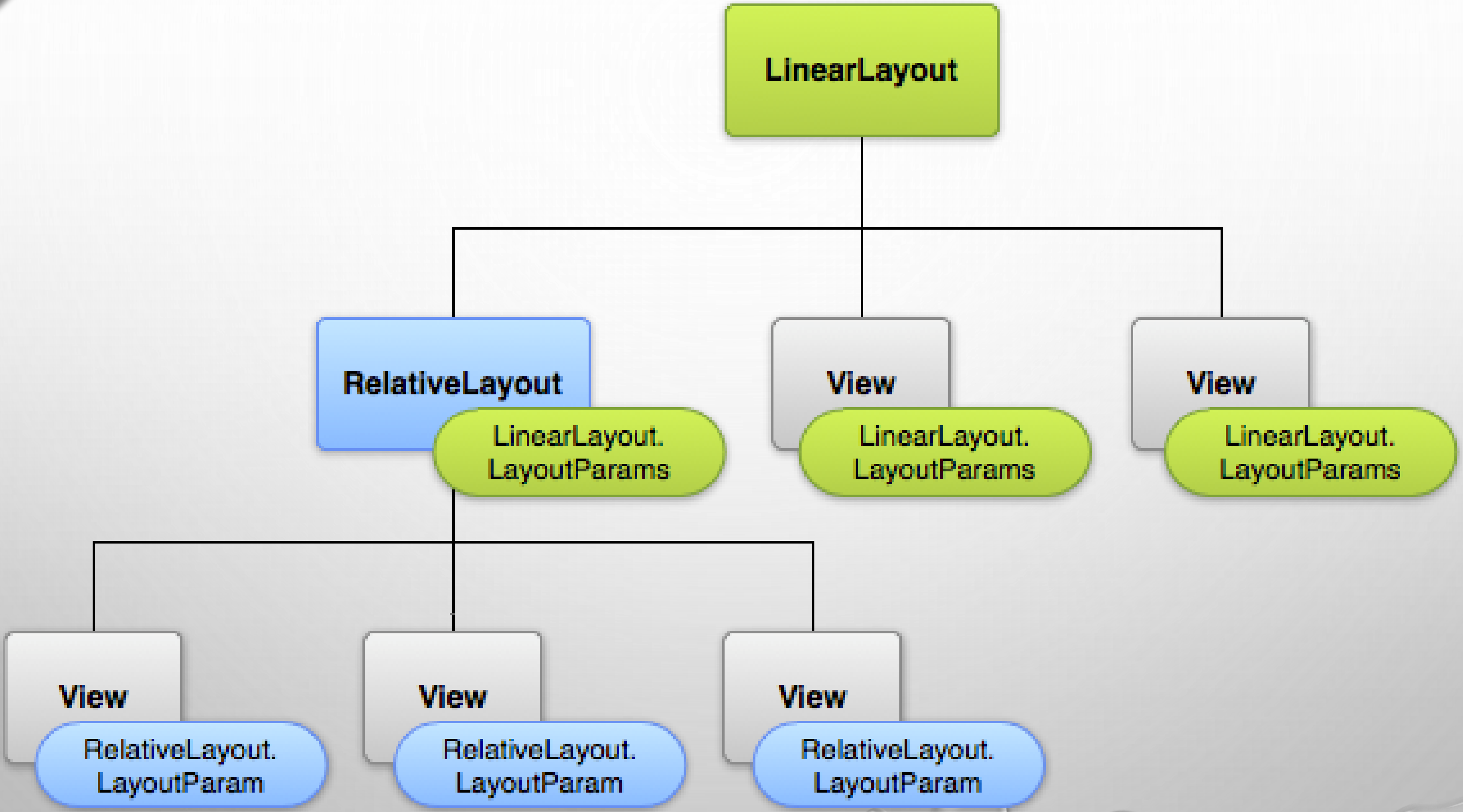
- THERE ARE A NUMBER OF OTHER ID RESOURCES THAT ARE OFFERED BY THE ANDROID FRAMEWORK.
- WHEN REFERENCING AN ANDROID RESOURCE ID, YOU DO NOT NEED THE PLUS-SYMBOL, BUT MUST ADD THE ANDROID PACKAGE NAMESPACE, LIKE SO:
  - **ANDROID:ID="@ANDROID:ID/EMPTY“**
- DEFINE A VIEW/WIDGET IN THE LAYOUT FILE AND ASSIGN IT A UNIQUE ID:

```
<BUTTON ANDROID:ID="@+ID/MY_BUTTON"
            ANDROID:LAYOUT_WIDTH="WRAP_CONTENT"
            ANDROID:LAYOUT_HEIGHT="WRAP_CONTENT"
            ANDROID:TEXT="@STRING/MY_BUTTON_TEXT"/>
```

- THEN CREATE AN INSTANCE OF THE VIEW OBJECT AND CAPTURE IT FROM THE LAYOUT (TYPICALLY IN THE ONCREATE() METHOD):
  - **BUTTON MYBUTTON = (BUTTON) FINDVIEWBYID(R.ID.MY_BUTTON);**

LAYOUT PARAMETERS

- XML LAYOUT ATTRIBUTES NAMED LAYOUT_SOMETHING DEFINE LAYOUT PARAMETERS FOR THE VIEW THAT ARE APPROPRIATE FOR THE VIEWGROUP IN WHICH IT RESIDES.
- EVERY VIEWGROUP CLASS IMPLEMENTS A NESTED CLASS THAT EXTENDS VIEWGROUP.LAYOUTPARAMS. THIS SUBCLASS CONTAINS PROPERTY TYPES THAT DEFINE THE SIZE AND POSITION FOR EACH CHILD VIEW, AS APPROPRIATE FOR THE VIEW GROUP.


LinearLayout
RelativeLayout
LinearLayout. LayoutParams
View
LinearLayout. LayoutParams
View
LinearLayout. LayoutParams
View
RelativeLayout. LayoutParam
View
RelativeLayout. LayoutParam
View
RelativeLayout. LayoutParam

CONT …

- NOTE THAT EVERY LAYOUTPARAMS SUBCLASS HAS ITS OWN SYNTAX FOR SETTING VALUES.
- EACH CHILD ELEMENT MUST DEFINE LAYOUTPARAMS THAT ARE APPROPRIATE FOR ITS PARENT, THOUGH IT MAY ALSO DEFINE DIFFERENT LAYOUTPARAMS FOR ITS OWN CHILDREN.
- ALL VIEW GROUPS INCLUDE A WIDTH AND HEIGHT (LAYOUT_WIDTH AND LAYOUT_HEIGHT), AND EACH VIEW IS REQUIRED TO DEFINE THEM.
- MANY LAYOUTPARAMS ALSO INCLUDE OPTIONAL MARGINS AND BORDERS.
- YOU CAN SPECIFY WIDTH AND HEIGHT WITH EXACT MEASUREMENTS, THOUGH YOU PROBABLY WON'T WANT TO DO THIS OFTEN.
- MORE OFTEN, YOU WILL USE ONE OF THESE CONSTANTS TO SET THE WIDTH OR HEIGHT:
    - **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.

CONT …

- IN GENERAL, SPECIFYING A LAYOUT WIDTH AND HEIGHT USING ABSOLUTE UNITS SUCH AS PIXELS IS NOT RECOMMENDED.
- INSTEAD, USING RELATIVE MEASUREMENTS SUCH AS DENSITY-INDEPENDENT PIXEL UNITS (***DP***), ***WRAP_CONTENT***, OR ***MATCH_PARENT***, IS A BETTER APPROACH, BECAUSE IT HELPS ENSURE THAT YOUR APPLICATION WILL DISPLAY PROPERLY ACROSS A VARIETY OF DEVICE SCREEN SIZES.

LAYOUT POSITION

- THE GEOMETRY OF A VIEW IS THAT OF A RECTANGLE.
- A VIEW HAS A LOCATION, EXPRESSED AS A PAIR OF *LEFT* AND *TOP* COORDINATES, AND TWO DIMENSIONS, EXPRESSED AS A WIDTH AND A HEIGHT.
- THE UNIT FOR LOCATION AND DIMENSIONS IS THE PIXEL.
- IT IS POSSIBLE TO RETRIEVE THE LOCATION OF A VIEW BY INVOKING THE METHODS **GETLEFT()** AND **GETTOP()**.
- THE FORMER RETURNS THE LEFT, OR X, COORDINATE OF THE RECTANGLE REPRESENTING THE VIEW.
- THE LATTER RETURNS THE TOP, OR Y, COORDINATE OF THE RECTANGLE REPRESENTING THE VIEW.

CONT …

- IN ADDITION, SEVERAL CONVENIENCE METHODS ARE OFFERED TO AVOID UNNECESSARY COMPUTATIONS, NAMELY **GETRIGHT()** AND **GETBOTTOM()**.
- THESE METHODS RETURN THE COORDINATES OF THE RIGHT AND BOTTOM EDGES OF THE RECTANGLE REPRESENTING THE VIEW. FOR INSTANCE, CALLING **GETRIGHT()** IS SIMILAR TO THE FOLLOWING COMPUTATION: **GETLEFT()** + **GETWIDTH()**

SIZE, PADDING AND MARGINS

- THE SIZE OF A VIEW IS EXPRESSED WITH A WIDTH AND A HEIGHT.
- A VIEW ACTUALLY POSSESS TWO PAIRS OF WIDTH AND HEIGHT VALUES.
- THE FIRST PAIR IS KNOWN AS MEASURED WIDTH AND MEASURED HEIGHT.
- THESE DIMENSIONS DEFINE HOW BIG A VIEW WANTS TO BE WITHIN ITS PARENT.
- THE MEASURED DIMENSIONS CAN BE OBTAINED BY CALLING GETMEASUREDWIDTH() AND GETMEASUREDHEIGHT().
- THE SECOND PAIR IS SIMPLY KNOWN AS WIDTH AND HEIGHT, OR SOMETIMES DRAWING WIDTH AND DRAWING HEIGHT.
- THESE DIMENSIONS DEFINE THE ACTUAL SIZE OF THE VIEW ON SCREEN, AT DRAWING TIME AND AFTER LAYOUT.

CONT …

- THESE VALUES MAY, BUT DO NOT HAVE TO, BE DIFFERENT FROM THE MEASURED WIDTH AND HEIGHT.
- THE WIDTH AND HEIGHT CAN BE OBTAINED BY CALLING GETWIDTH() AND GETHEIGHT().
- TO MEASURE ITS DIMENSIONS, A VIEW TAKES INTO ACCOUNT ITS PADDING.
- THE PADDING IS EXPRESSED IN PIXELS FOR THE LEFT, TOP, RIGHT AND BOTTOM PARTS OF THE VIEW.
- PADDING CAN BE USED TO OFFSET THE CONTENT OF THE VIEW BY A SPECIFIC NUMBER OF PIXELS.

CONT …

- FOR INSTANCE, A LEFT PADDING OF 2 WILL PUSH THE VIEW'S CONTENT BY 2 PIXELS TO THE RIGHT OF THE LEFT EDGE.
- PADDING CAN BE SET USING THE SETPADDING(INT, INT, INT, INT) METHOD AND QUERIED BY CALLING GETPADDINGLEFT(), GETPADDINGTOP(), GETPADDINGRIGHT() AND GETPADDINGBOTTOM().
- EVEN THOUGH A VIEW CAN DEFINE A PADDING, IT DOES NOT PROVIDE ANY SUPPORT FOR MARGINS. HOWEVER, VIEW GROUPS PROVIDE SUCH A SUPPORT.
- REFER TO VIEWGROUP AND VIEWGROUP.MARGINLAYOUTPARAMS FOR FURTHER INFORMATION

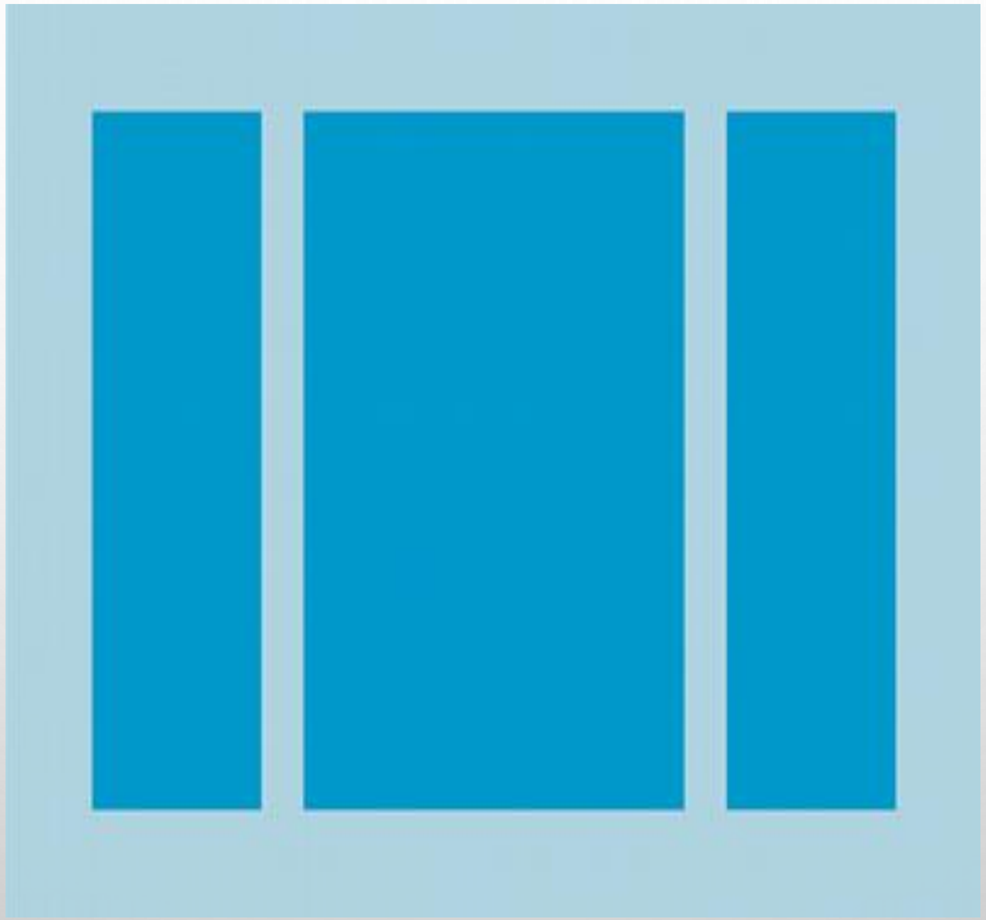
COMMON LAYOUTS

- ALTHOUGH YOU CAN NEST ONE OR MORE LAYOUTS WITHIN ANOTHER LAYOUT TO ACHEIVE YOUR UI DESIGN, YOU SHOULD STRIVE TO KEEP YOUR LAYOUT HIERARCHY AS SHALLOW AS POSSIBLE.
- YOUR LAYOUT DRAWS FASTER IF IT HAS FEWER NESTED LAYOUTS (A WIDE VIEW HIERARCHY IS BETTER THAN A DEEP VIEW HIERARCHY).

LINEAR LAYOUT

- LINEARLAYOUT IS A VIEW GROUP THAT ALIGNS ALL CHILDREN IN A SINGLE DIRECTION, VERTICALLY OR HORIZONTALLY.
- YOU CAN SPECIFY THE LAYOUT DIRECTION WITH THE ANDROID:ORIENTATION ATTRIBUTE.
- ALL CHILDREN OF A LINEARLAYOUT ARE STACKED ONE AFTER THE OTHER, SO A VERTICAL LIST WILL ONLY HAVE ONE CHILD PER ROW, NO MATTER HOW WIDE THEY ARE, AND A HORIZONTAL LIST WILL ONLY BE ONE ROW HIGH (THE HEIGHT OF THE TALLEST CHILD, PLUS PADDING).
- A LINEARLAYOUT RESPECTS MARGINS BETWEEN CHILDREN AND THE GRAVITY (RIGHT, CENTER, OR LEFT ALIGNMENT) OF EACH CHILD.

CONT …



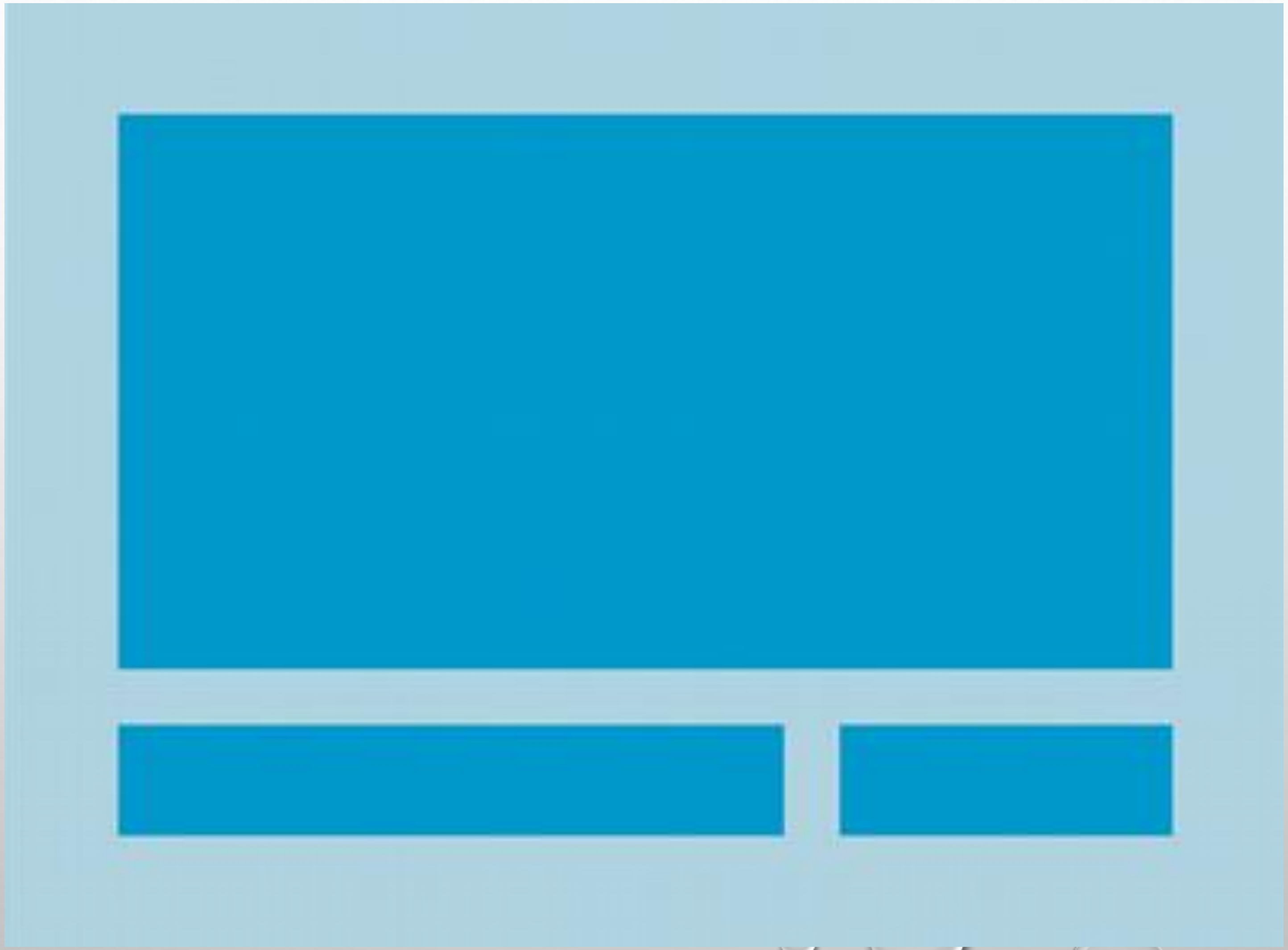
CONT …

- LAYOUT WEIGHT
  - LINEARLAYOUT ALSO SUPPORTS ASSIGNING A WEIGHT TO INDIVIDUAL CHILDREN WITH THE **ANDROID:LAYOUT_WEIGHT** ATTRIBUTE.
  - THIS ATTRIBUTE ASSIGNS AN "IMPORTANCE" VALUE TO A VIEW IN TERMS OF HOW MUCH SPACE IT SHOULD OCCUPY ON THE SCREEN.
  - A LARGER WEIGHT VALUE ALLOWS IT TO EXPAND TO FILL ANY REMAINING SPACE IN THE PARENT VIEW.
  - CHILD VIEWS CAN SPECIFY A WEIGHT VALUE, AND THEN ANY REMAINING SPACE IN THE VIEW GROUP IS ASSIGNED TO CHILDREN IN THE PROPORTION OF THEIR DECLARED WEIGHT.
  - DEFAULT WEIGHT IS ZERO

RELATIVE LAYOUT

- RELATIVELAYOUT IS A VIEW GROUP THAT DISPLAYS CHILD VIEWS IN RELATIVE POSITIONS.
- THE POSITION OF EACH VIEW CAN BE SPECIFIED AS RELATIVE TO SIBLING ELEMENTS (SUCH AS TO THE LEFT-OF OR BELOW ANOTHER VIEW) OR IN POSITIONS RELATIVE TO THE PARENT RELATIVELAYOUT AREA (SUCH AS ALIGNED TO THE BOTTOM, LEFT OR CENTER).
- A RELATIVELAYOUT IS A VERY POWERFUL UTILITY FOR DESIGNING A USER INTERFACE BECAUSE IT CAN ELIMINATE NESTED VIEW GROUPS AND KEEP YOUR LAYOUT HIERARCHY FLAT, WHICH IMPROVES PERFORMANCE.
- IF YOU FIND YOURSELF USING SEVERAL NESTED LINEARLAYOUT GROUPS, YOU MAY BE ABLE TO REPLACE THEM WITH A SINGLE RELATIVELAYOUT

CONT …



CONT …

- POSITIONING VIEWS
  - RELATIVELAYOUT LETS CHILD VIEWS SPECIFY THEIR POSITION RELATIVE TO THE PARENT VIEW OR TO EACH OTHER (SPECIFIED BY ID).
  - SO YOU CAN ALIGN TWO ELEMENTS BY RIGHT BORDER, OR MAKE ONE BELOW ANOTHER, CENTERED IN THE SCREEN, CENTERED LEFT, AND SO ON.
  - BY DEFAULT, ALL CHILD VIEWS ARE DRAWN AT THE TOP-LEFT OF THE LAYOUT, SO YOU MUST DEFINE THE POSITION OF EACH VIEW USING THE VARIOUS LAYOUT PROPERTIES AVAILABLE FROM RELATIVELAYOUT.LAYOUTPARAMS.
  - SOME OF THE MANY LAYOUT PROPERTIES AVAILABLE TO VIEWS IN A RELATIVELAYOUT INCLUDE:
    - **ANDROID:LAYOUT_ALIGNPARENTTOP**: IF "TRUE", MAKES THE TOP EDGE OF THIS VIEW MATCH THE TOP EDGE OF THE PARENT.
    - **ANDROID:LAYOUT_CENTERVERTICAL**: IF "TRUE", CENTERS THIS CHILD VERTICALLY WITHIN ITS PARENT.
    - **ANDROID:LAYOUT_BELOW**: POSITIONS THE TOP EDGE OF THIS VIEW BELOW THE VIEW SPECIFIED WITH A RESOURCE ID.
    - **ANDROID:LAYOUT_TORIGHTOF**: POSITIONS THE LEFT EDGE OF THIS VIEW TO THE RIGHT OF THE VIEW SPECIFIED WITH A RESOURCE ID.
    - THE REST CAN BE FOUND AT HTTPS://DEVELOPER.ANDROID.COM/REFERENCE/ANDROID/WIDGET/RELATIVELAYOUT.LAYOUTPARAMS.HTML

RECYCLER VIEW

- MANY APPS NEED TO DISPLAY USER-INTERFACE ELEMENTS BASED ON LARGE DATA SETS, OR DATA THAT FREQUENTLY CHANGES.
- FOR EXAMPLE, A MUSIC APP MIGHT NEED TO DISPLAY INFORMATION ABOUT THOUSANDS OF ALBUMS, BUT ONLY A DOZEN OF THOSE ALBUMS MIGHT BE ON-SCREEN AT A TIME.
- IF THE APP CREATED UI WIDGETS FOR EACH OF THOSE ALBUMS, THE APP WOULD END UP USING A LOT OF MEMORY AND STORAGE, POTENTIALLY MAKING THE APP SLOW AND CRASH-PRONE.
- ON THE OTHER HAND, IF THE APP CREATED UI WIDGETS EACH TIME A NEW ALBUM SCROLLED ONTO THE SCREEN AND DESTROYED THE WIDGETS WHEN IT SCROLLED OFF, THAT WOULD ALSO CAUSE THE APP TO RUN SLOWLY, SINCE CREATING UI OBJECTS IS A RESOURCE-INTENSIVE OPERATION.

CONT …

- TO ADDRESS THIS COMMON SITUATION, THE ANDROID SUPPORT LIBRARY PROVIDES THE RECYCLERVIEW SUITE OF OBJECTS.
- RECYCLERVIEW AND ITS ASSOCIATED CLASSES AND INTERFACES HELP YOU TO DESIGN AND IMPLEMENT A DYNAMIC USER INTERFACE THAT RUNS EFFICIENTLY.
- HTTPS://DEVELOPER.ANDROID.COM/GUIDE/TOPICS/UI/LAYOUT/RECYCLERVIEW.HTML

LIST VIEW

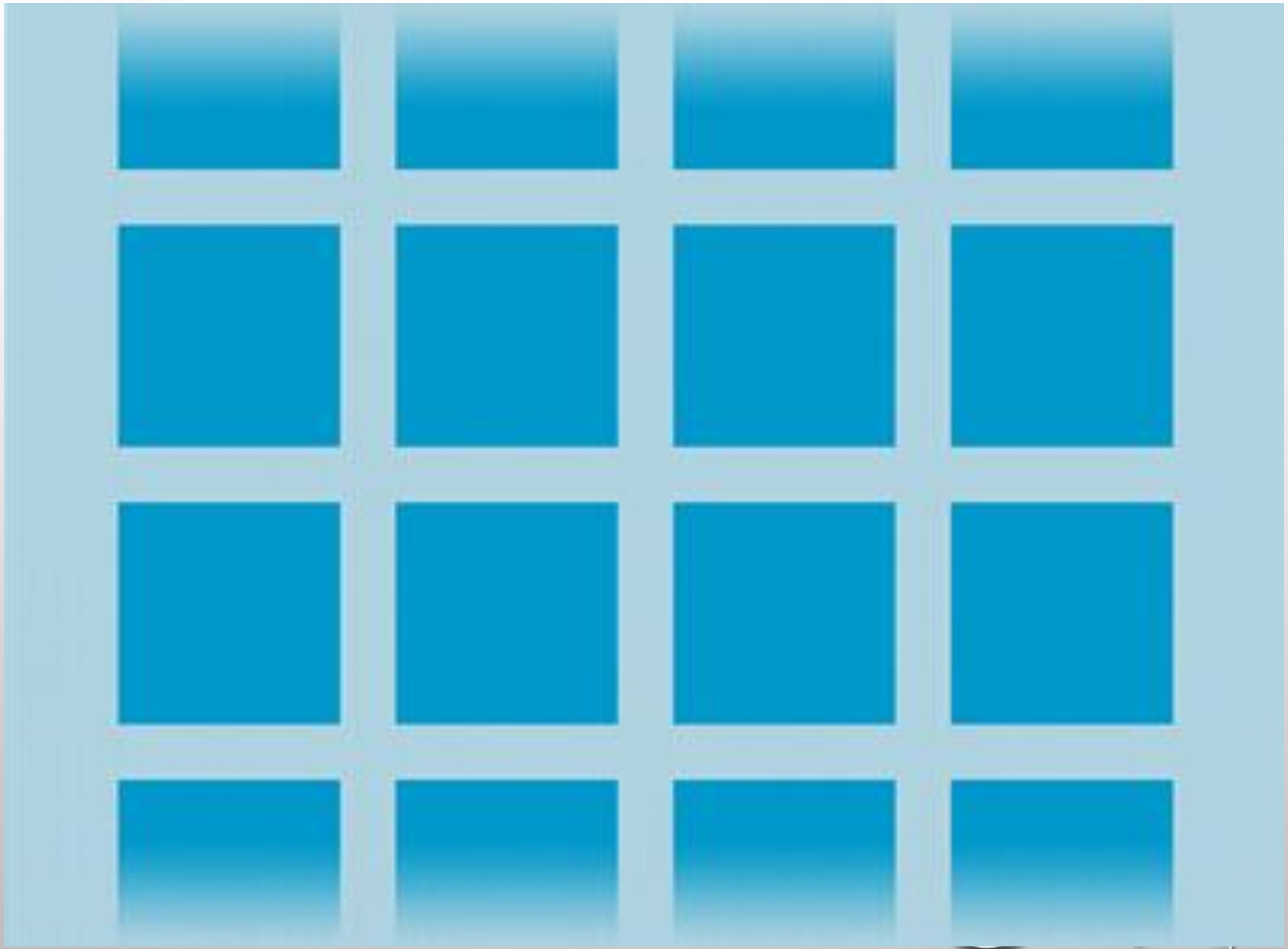
- LISTVIEW IS A VIEW GROUP THAT DISPLAYS A LIST OF SCROLLABLE ITEMS.
- THE LIST ITEMS ARE AUTOMATICALLY INSERTED TO THE LIST USING AN ADAPTER THAT PULLS CONTENT FROM A SOURCE SUCH AS AN ARRAY OR DATABASE QUERY AND CONVERTS EACH ITEM RESULT INTO A VIEW THAT'S PLACED INTO THE LIST.

CONT …

GRID VIEW

- GRIDVIEW IS A VIEWGROUP THAT DISPLAYS ITEMS IN A TWO-DIMENSIONAL, SCROLLABLE GRID.
- THE GRID ITEMS ARE AUTOMATICALLY INSERTED TO THE LAYOUT USING A LISTADAPTER.

CONT …



INPUT CONTROLS

- ANDROID PROVIDES A WIDE VARIETY OF CONTROLS YOU CAN USE IN YOUR UI, SUCH AS BUTTONS, TEXT FIELDS, SEEK BARS, CHECKBOXES, ZOOM BUTTONS, TOGGLE BUTTONS, AND MANY MORE.
- ADDING AN INPUT CONTROL TO YOUR UI IS AS SIMPLE AS ADDING AN XML ELEMENT TO YOUR XML LAYOUT.
- **COMMON CONTROLS**
  - **BUTTON**: A PUSH-BUTTON THAT CAN BE PRESSED, OR CLICKED, BY THE USER TO PERFORM AN ACTION.
  - **TEXT FIELD**: AN EDITABLE TEXT FIELD. YOU CAN USE THE AUTOCOMPLETETEXTVIEW WIDGET TO CREATE A TEXT ENTRY WIDGET THAT PROVIDES AUTO-COMPLETE SUGGESTIONS
  - **CHECKBOX**: AN ON/OFF SWITCH THAT CAN BE TOGGLED BY THE USER. YOU SHOULD USE CHECKBOXES WHEN PRESENTING USERS WITH A GROUP OF SELECTABLE OPTIONS THAT ARE NOT MUTUALLY EXCLUSIVE.
  - **RADIO BUTTON:** SIMILAR TO CHECKBOXES, EXCEPT THAT ONLY ONE OPTION CAN BE SELECTED IN THE GROUP.
  - TOGGLE BUTTON: AN ON/OFF BUTTON WITH A LIGHT INDICATOR.
  - SPINNER: A DROP-DOWN LIST THAT ALLOWS USERS TO SELECT ONE VALUE FROM A SET.
  - PICKERS: A DIALOG FOR USERS TO SELECT A SINGLE VALUE FOR A SET BY USING UP/DOWN BUTTONS OR VIA A SWIPE GESTURE. USE A DATEPICKERCODE> WIDGET TO ENTER THE VALUES FOR THE DATE (MONTH, DAY, YEAR) OR A TIMEPICKER WIDGET TO ENTER THE VALUES FOR A TIME (HOUR, MINUTE, AM/PM), WHICH WILL BE FORMATTED AUTOMATICALLY FOR THE USER'S LOCALE.

CONT …

- HTTPS://DEVELOPER.ANDROID.COM/GUIDE/TOPICS/UI/CONTROLS/BUTTON.HTML
- HTTPS://DEVELOPER.ANDROID.COM/GUIDE/TOPICS/UI/CONTROLS/CHECKBOX.HTML
- HTTPS://DEVELOPER.ANDROID.COM/GUIDE/TOPICS/UI/CONTROLS/RADIOBUTTON.HTML
- HTTPS://DEVELOPER.ANDROID.COM/GUIDE/TOPICS/UI/CONTROLS/TOGGLEBUTTON.HTML
- HTTPS://DEVELOPER.ANDROID.COM/GUIDE/TOPICS/UI/CONTROLS/SPINNER.HTML
- HTTPS://DEVELOPER.ANDROID.COM/GUIDE/TOPICS/UI/CONTROLS/PICKERS.HTML