



# HUST

**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**  
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.



TRƯỜNG ĐẠI HỌC  
BÁCH KHOA HÀ NỘI  
HANOI UNIVERSITY  
OF SCIENCE AND TECHNOLOGY

# LẬP TRÌNH ỨNG DỤNG DI ĐỘNG

## Mobile Application Programming

ET4710

PGS. TS. Đỗ Trọng Tuấn

Viện Điện tử Viễn thông \* Đại học Bách Khoa Hà Nội

ONE LOVE. ONE FUTURE.

## CHƯƠNG 4.

### Thiết kế và lập trình giao diện người dùng (**User Interface Design and Programming**)



## Chương 4

### Thiết kế và lập trình giao diện người dùng (**User Interface Design and Programming**)

4.1. Thiết kế giao diện UI với Layouts và Widgets

(**Creating UI with Layouts and Widgets**)

4.2 Các công cụ đồ họa trong giao diện người dùng

(**Graphical UI tools**)

## User Interfaces

- **UI** (User interface) is **everything** that the user can **see** and **interact** with.
- Android provides a variety of **pre-built UI components** such as **structured layout objects** and **UI controls** that allow you to build the graphical user interface for your app.
- Android also provides other **UI modules** for special interfaces such as dialogs, notifications, and menus.
- The user interface (**UI**) for an Android app is built as a **hierarchy of layouts** and **widgets**.

# Thiết kế giao diện UI với Layouts và Widgets

## User Interfaces

- Android provides an **XML vocabulary** for **ViewGroup** and **View classes**, so **most** of your **UI** is **defined** in **XML files**.

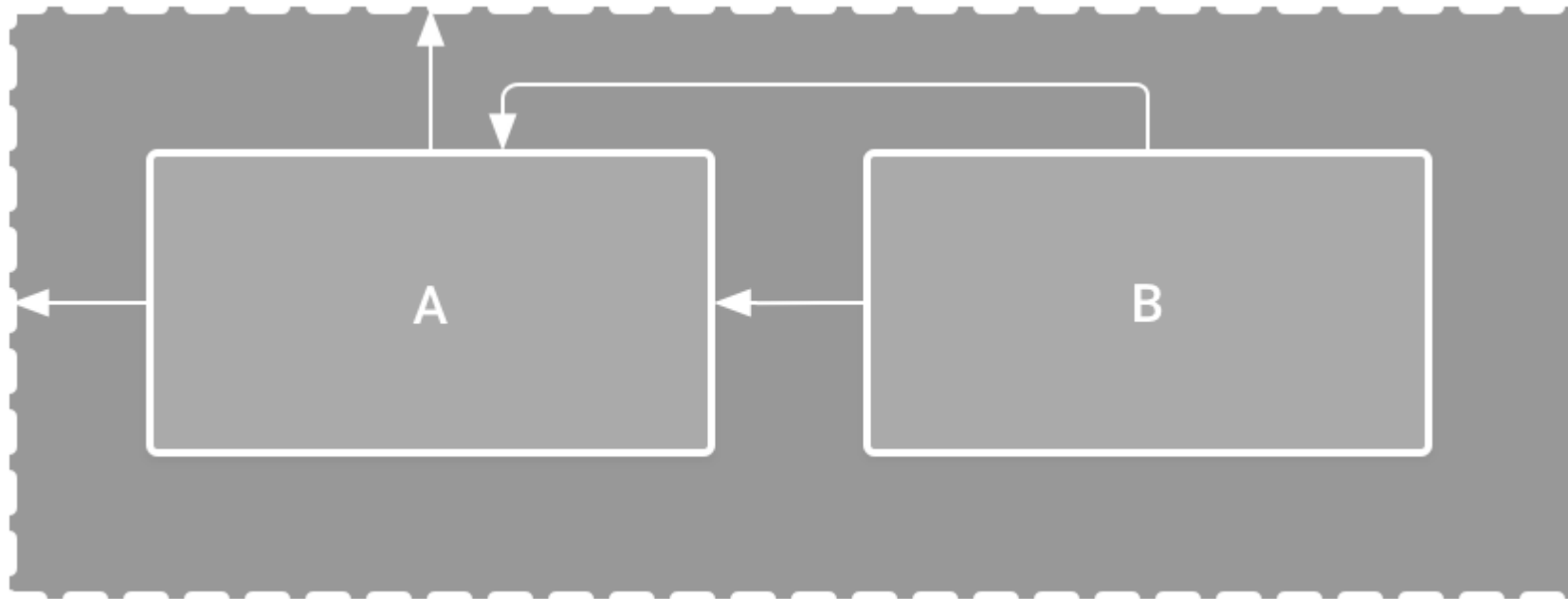
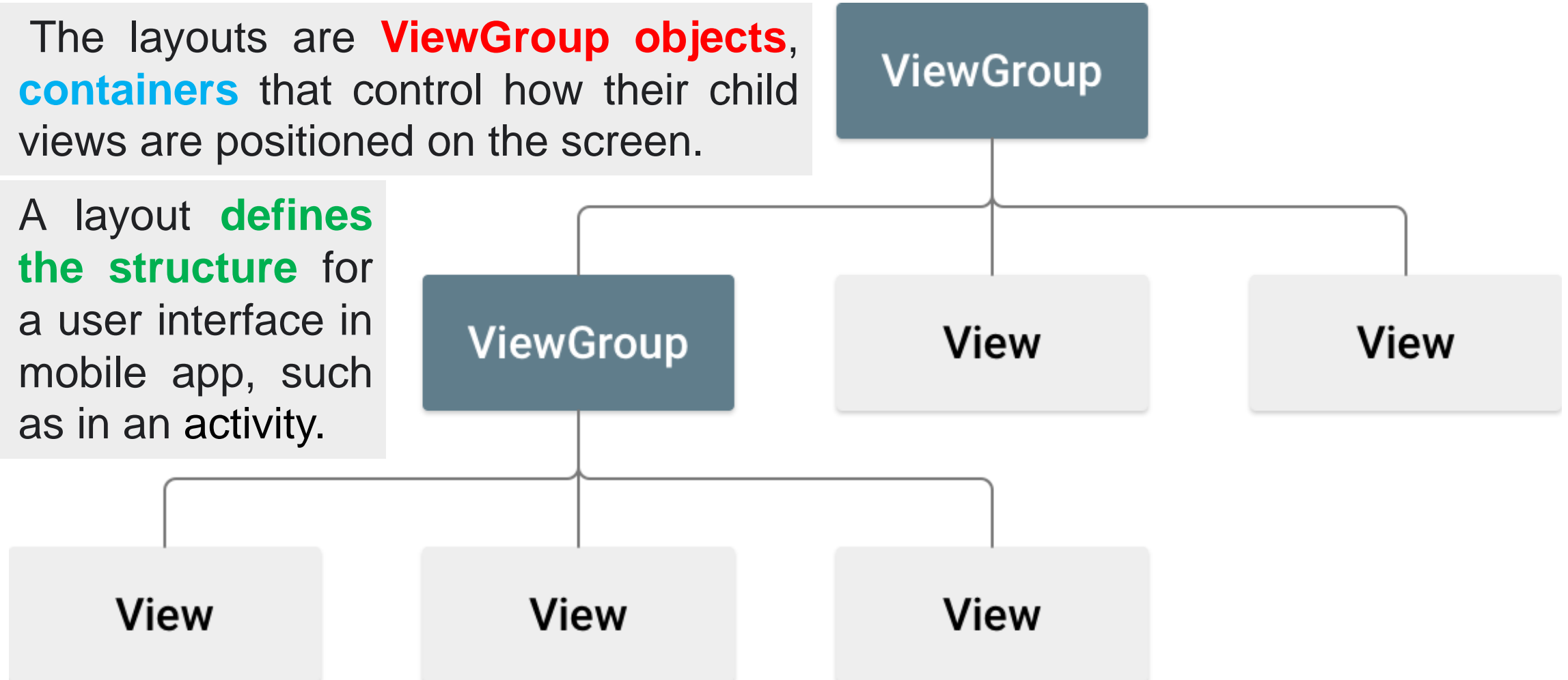


Illustration of two **views positioned inside ConstraintLayout**

# Thiết kế giao diện UI với Layouts và Widgets

## Layouts

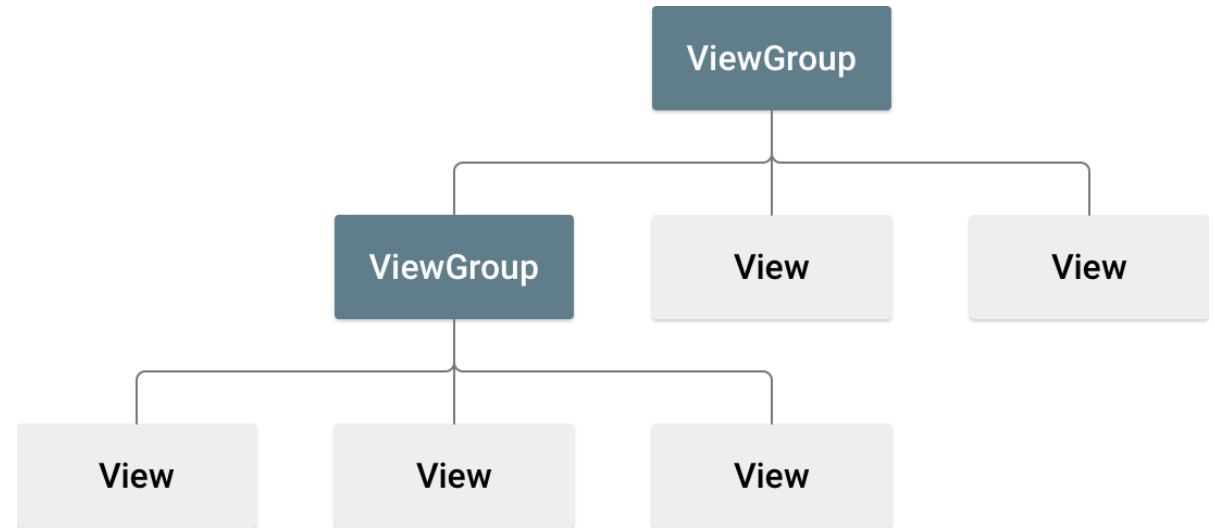
- The layouts are **ViewGroup objects**, **containers** that control how their child views are positioned on the screen.
- A layout **defines the structure** for a user interface in mobile app, such as in an activity.



# Thiết kế giao diện UI với Layouts và Widgets

## Layouts

A layout defines the structure for a user interface in mobile app, such as in an activity.



- All **elements** in the layout are built using a hierarchy of **View** and **ViewGroup** objects. A **View** usually *draws something the user can see and interact with*.
- Whereas a **ViewGroup** is an *invisible container* that defines the *layout structure* for View and other ViewGroup objects.
- **Widgets** are **View objects**, **UI components** such as *buttons* and *text boxes*.



# Thiết kế giao diện UI với Layouts và Widgets

## Layout Editor

The screenshot shows the Android Studio interface for editing a layout. A central design preview shows two rectangular views, one light gray and one teal. A white callout box with numbered points 1 through 7 explains the interface components:

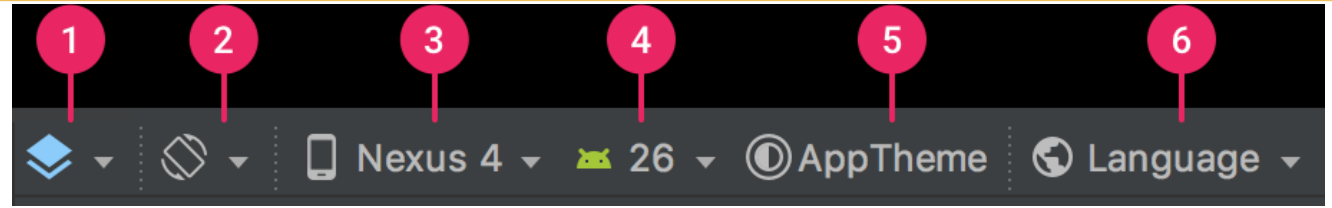
- 1 **Palette:** Contains various views and view groups that you can drag into your layout.
- 2 **Component Tree:** Shows the hierarchy of components in your layout.
- 3 **Toolbar:** Click these buttons to configure your layout appearance in the editor and change layout attributes.
- 4 **Design editor:** Edit your layout in Design view, Blueprint view, or both.
- 5 **Attributes:** Controls for the selected view's attributes.
- 6 **View mode:** View your layout in either **Code**, **Design**, or **Split** modes. **Split** mode shows both the **Code** and **Design** windows at the same time.
- 7 **Zoom and pan controls:** Control the preview size and position within the editor.

The interface also shows the **Resource Manager** on the left with a list of widgets like **TextView**, **Button**, **RecyclerView**, etc. The **Attributes** panel on the right shows settings for the selected **my\_button**, including **Declared Attributes**, **Layout**, **Constraint Widget**, and **Common Attributes**.



# Thiết kế giao diện UI với Layouts và Widgets

- **Configure the appearance of the layout in the editor**






- 1 Design and blueprint:** Select how you'd like to view your layout in the editor. Choose **Design** to see a rendered preview of your layout. Choose **Blueprint** to see only outlines for each view. Choose **Design + Blueprint** to see both views side-by-side. You can also press **B** to cycle through these view types.
- 2 Screen orientation and layout variants:** Choose between landscape and portrait screen orientation, or choose other screen modes for which your app provides alternative layouts, such as night mode. This menu also contains commands for [creating a new layout variant](#). You can also press **O** to change orientation.
- 3 Device type and size:** Select the device type (phone/tablet, Android TV, or Wear OS) and screen configuration (size and density). You can select from several pre-configured device types and your own AVD definitions, or you can create a new AVD by selecting **Add Device Definition** from the list. You can resize the device size by dragging the bottom-right corner of the layout. You can also press **D** to cycle through the device list.
- 4 API version:** Select the version of Android on which to preview your layout.
- 5 App theme:** Select which UI theme to apply to the preview. Note that this works only for supported layout styles, so many themes in this list result in an error.
- 6 Language:** Select the language to show for your UI strings. This list displays only the languages available in your string resources. If you'd like to edit your translations, click **Edit Translations** from the drop-down menu.

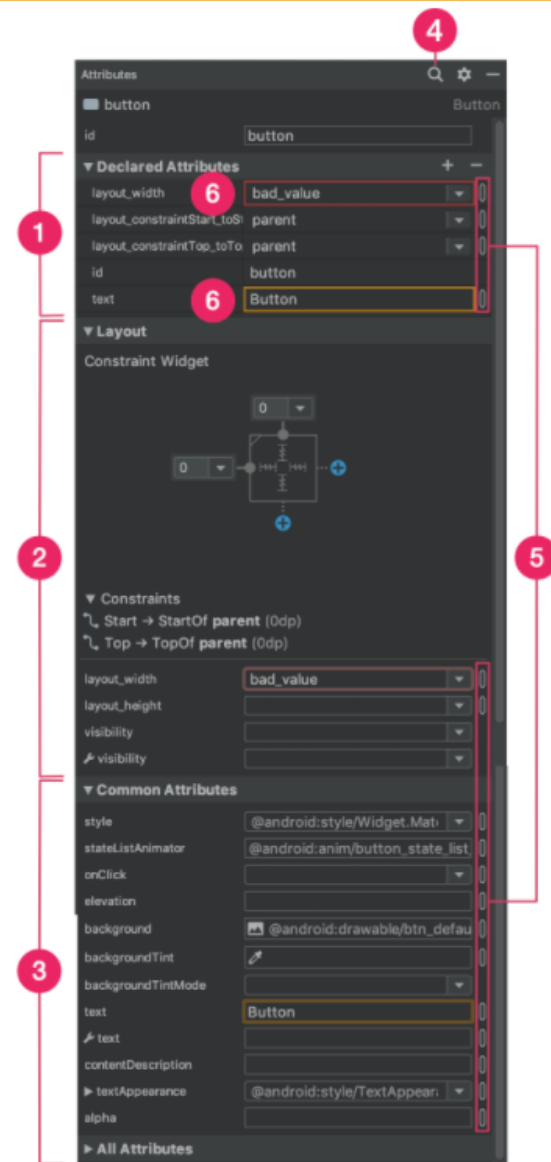
# Thiết kế giao diện UI với Layouts và Widgets

## • Edit view attributes

- You can edit view attributes from the Attributes window on the right side of the Layout Editor. This window is available only when the design editor is open, so be sure you're using either Design or Split mode to view your layout.
- When you select a view, whether by clicking the view in the Component Tree or in the design editor, the Attributes window shows



- 1 The **Declared Attributes** section lists attributes specified in the layout file. To add an attribute, click the **Add**  button at the top right of the section.
- 2 The **Layout** section contains controls for the width and height of the view. If the view is in a `ConstraintLayout`, this section also shows constraint bias and lists the constraints that the view uses. For more information on working with `ConstraintLayout`, see [Build a Responsive UI with ConstraintLayout](#).
- 3 The **Common Attributes** section lists common attributes for the selected view. To see all available attributes, expand the **All Attributes** section at the bottom of the window.
- 4 Click the **Search** button to search for a specific view attribute.
- 5 The icons to the right of each attribute value indicate whether the attribute values are resource references. These indicators are solid  when the value is a resource reference and empty  when the value is hard-coded. These indicators help you recognize hard-coded values at a glance. Clicking indicators in either state opens the **Resources** dialog window where you can select a resource reference for the corresponding attribute.
- 6 A red highlight around an attribute value indicates an error with the value. An error might indicate an invalid entry for a layout-defining attribute,



# Thiết kế giao diện UI với Layouts và Widgets

## Write the XML

- Using Android's **XML vocabulary**, you can quickly design **UI layouts** and the **screen elements** they contain with a *series of nested elements*.
- Each layout file must contain exactly one **root element**, which must be a **View** or **ViewGroup object**.
- Once you've defined the root element, you can add **additional layout objects** or **widgets** as *child elements* to gradually build a **View hierarchy** that defines your layout.

An **XML layout** that uses a **vertical LinearLayout** to hold a **TextView** and a **Button**

```
<?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" >
    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />
</LinearLayout>
```

After you've **declared your layout in XML**, **save the file with the .xml extension**, in your Android project's **res/layout/ directory**, so it will **properly compile**.

# Thiết kế giao diện UI với Layouts và Widgets

## Load the XML Resource

- **Compile** your app, each **XML layout file** is compiled into a **View resource**.
- **Load the layout resource** from your app code, in your **Activity.onCreate()** callback implementation.
- Do so by calling **setContentView()**, passing it the reference to your layout resource in the form of: **R.layout.layout\_file\_name**.
- The onCreate() callback method in your Activity is called by the Android framework when your Activity is launched

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.main_layout);  
}
```

# Thiết kế giao diện UI với Layouts và Widgets

## 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).
- 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.



# Thiết kế giao diện UI với Layouts và Widgets

## View ID (IDentify)

- Any View object may have an integer ID associated with it, to **uniquely identify** the **View** within the **tree**.
- ID is referenced as an **integer after being complied**, but the ID is **typically assigned** as a **string** in the layout XML file.
- This is an XML attribute common to all View objects (defined by the View class) and you will use it very often. 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**).

## Create and reference Views

- 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" />
```

- Create an instance of the view object and capture it from the layout (typically in the onCreate() method):

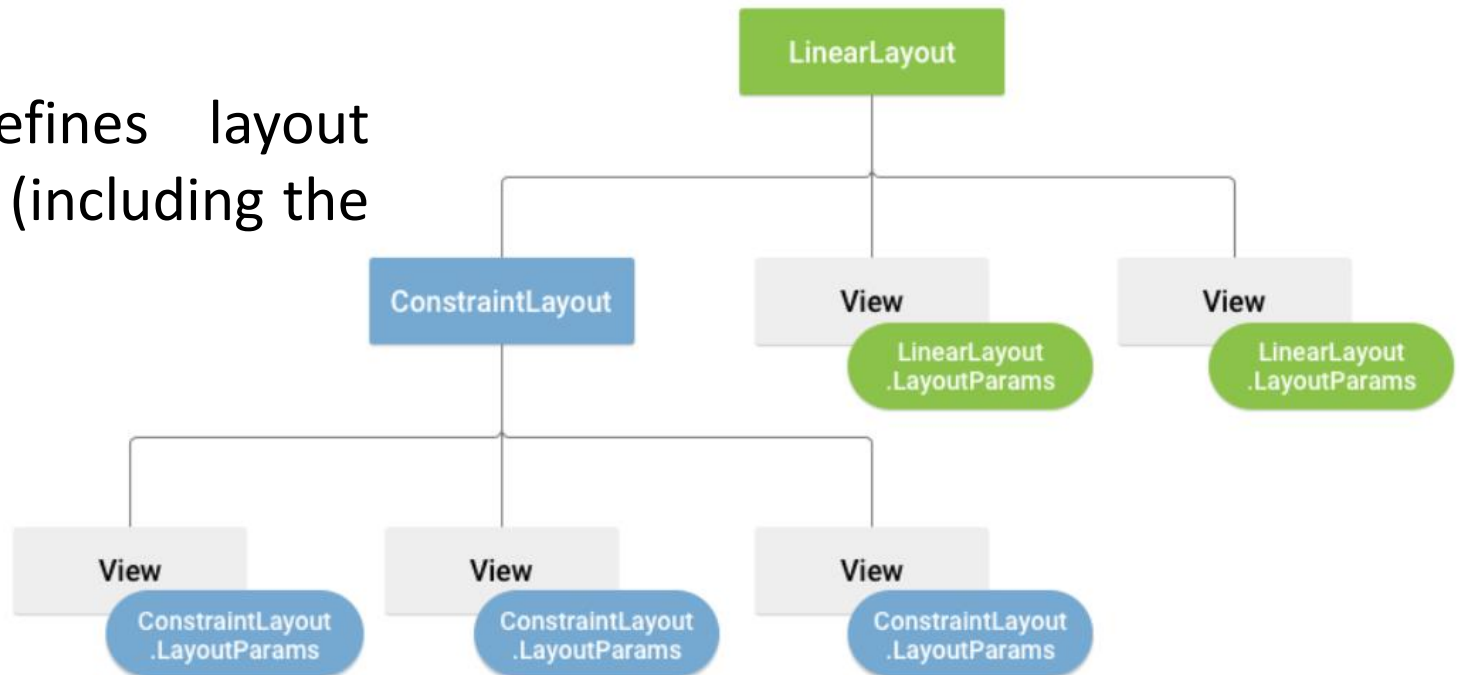
```
Button myButton = (Button) findViewById(R.id.my_button);
```



# Thiết kế giao diện UI với Layouts và Widgets

## 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 containing property types that define the **size and position** for each child view.
- The parent view group defines layout parameters for each child view (including the child view group).
- Each child element must define LayoutParams that are appropriate for its parent.



# Thiết kế giao diện UI với Layouts và Widgets

## Layout Parameters

- 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** or 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.
- 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 app will **display properly across a variety of device screen sizes**.



# Thiết kế giao diện UI với Layouts và Widgets

## 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()**, both return the location of the view **relative to its parent**.
- For instance, when **getLeft() returns 20**, that means the view is located 20 pixels to the right of the left edge of its direct parent.
- In addition, several convenience methods are offered to avoid unnecessary computations, namely **getRight()** and **getBottom()**.
- For instance, calling **getRight()** is similar to the following computation: **getLeft() + getWidth()**.

# Thiết kế giao diện UI với Layouts và Widgets

## Size, Padding and Margins

- The **size of a view** is expressed with **a width** and **a height**.
- A view actually possesses 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. The width and height can be obtained by calling **getWidth()** and **getHeight()**.
- The **padding** is expressed in pixels for the left, top, right and bottom parts of the view:
- **setPadding(int, int, int, int)** ; **getPaddingLeft()**, **getPaddingTop()**, **getPaddingRight()** , **getPaddingBottom()**

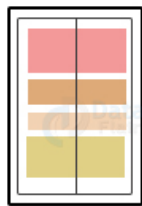


# Thiết kế giao diện UI với Layouts và Widgets

## Common Layouts

- Each subclass of the ViewGroup class provides a unique way to display the views you nest within it.

### Types of Android Layouts



StackLayout



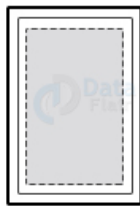
AbsoluteLayout



RelativeLayout



GridLayout



ContentView



ScrollView



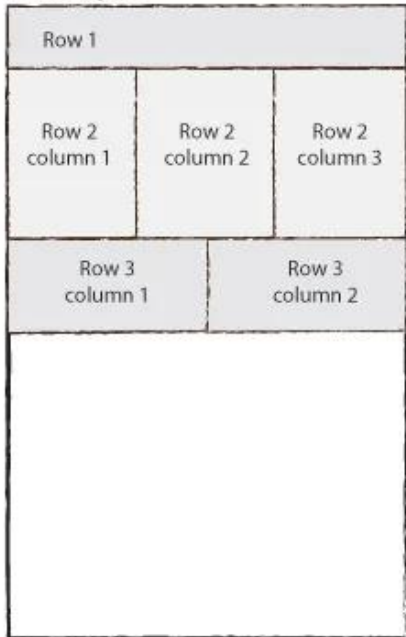
Frame

Sr.No	Layout & Description
1	Linear Layout <a href="#">↗</a> LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally.
2	Relative Layout <a href="#">↗</a> RelativeLayout is a view group that displays child views in relative positions.
3	Table Layout <a href="#">↗</a> TableLayout is a view that groups views into rows and columns.
4	Absolute Layout <a href="#">↗</a> AbsoluteLayout enables you to specify the exact location of its children.
5	Frame Layout <a href="#">↗</a> The FrameLayout is a placeholder on screen that you can use to display a single view.
6	List View <a href="#">↗</a> ListView is a view group that displays a list of scrollable items.
7	Grid View <a href="#">↗</a> GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid.

# Thiết kế giao diện UI với Layouts và Widgets

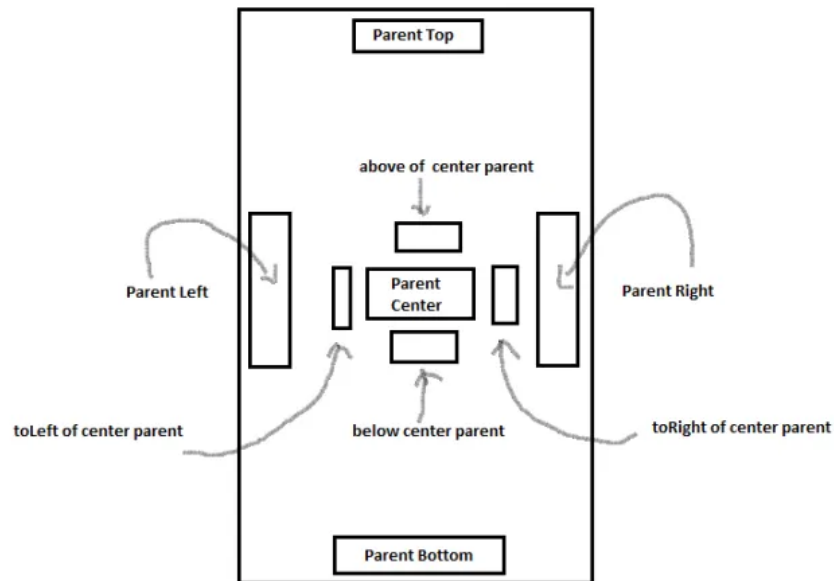
## Common Layouts

<TableLayout>



</TableLayout>

TableLayout



RelativeLayout

LinearLayout



Grid View



## Building Layouts with an Adapter

- When the **content** for your layout is **dynamic** or **not pre-determined**, you can use a layout that subclasses **AdapterView** to *populate the layout* with views **at runtime**.
- A subclass of the AdapterView class **uses** an **Adapter** to **bind data** to its **layout**.
- The Adapter behaves as a **middleman** between the **data source** and the **AdapterView layout**—the Adapter retrieves the data (from a source such as an array or a database query) and converts each entry into a view that can be added into the AdapterView layout.

## Building Layouts with an Adapter

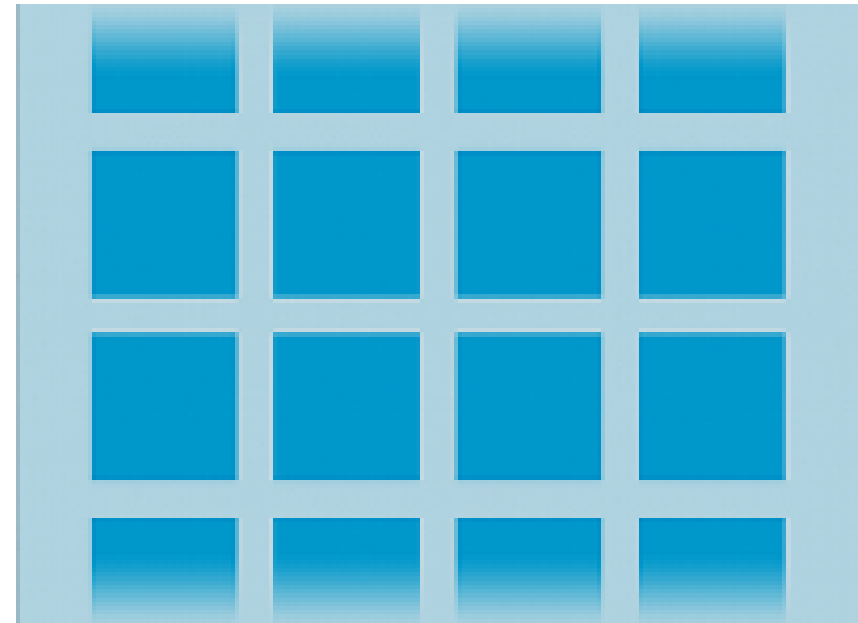
### List View

Displays a scrolling single column list



### Grid View

Displays a scrolling grid of columns and rows.



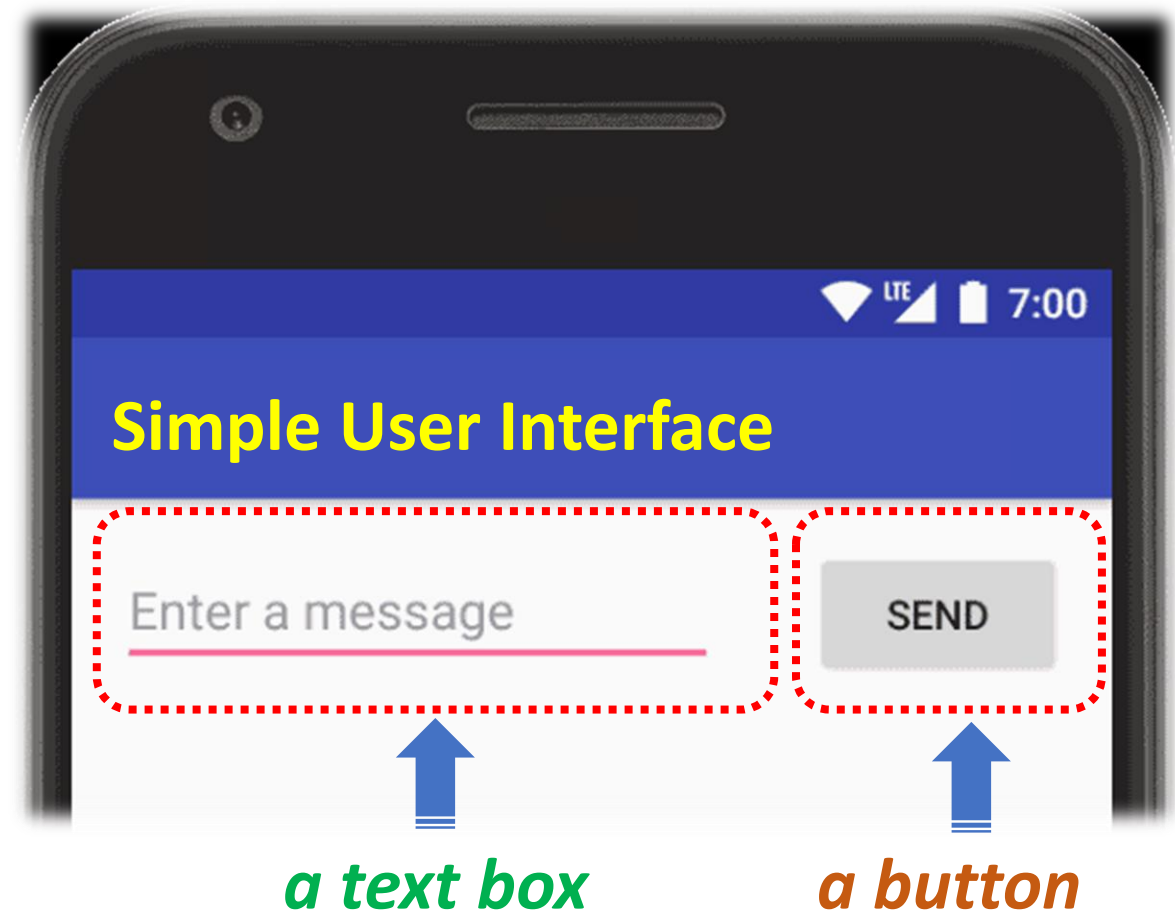
*Common layouts backed by an adapter*



# Thiết kế giao diện UI với Layouts và Widgets






## Build a simple user interface

- Use the **Android Studio Layout Editor** to create a layout that includes *a text box* and *a button*.



# Thiết kế giao diện UI với Layouts và Widgets

## Build a simple user interface

1. In the Project window, open **app > res > layout > activity\_main.xml**.
2. To make room for the Layout Editor, hide the **Project** window. To do so, select **View > Tool Windows > Project**, or just click **Project** on the left side of the Android Studio screen.
3. If your editor shows the XML source, click the **Design** tab at the top right of the window.
4. Click  (**Select Design Surface**) and select **Blueprint**.
5. Click  (**View Options**) in the Layout Editor toolbar and make sure that **Show All Constraints** is checked.
6. Make sure Autoconnect is off. A tooltip in the toolbar displays  (**Enable Autoconnection to Parent**) when Autoconnect is off.
7. Click  (**Default Margins**) in the toolbar and select **16**. If needed, you can adjust the margins for each view later.
8. Click  (**Device for Preview**) in the toolbar and select **5.5, 1440 × 2560, 560 dpi (Pixel XL)**.

## User Interface Testing

- Perform all user UI actions with views
  - Tap a UI view, and enter data or make a choice
  - Examine the values of the properties of each view
- Provide input to all UI views
  - Try invalid values
- Check returned output
  - Correct or expected values?
  - Correct presentation?

## Homework

### Build a Responsive UI with ConstraintLayout

<https://developer.android.com/training/constraint-layout>



**HUST**

**THANK YOU !**

# Lập trình ứng dụng di động

# Mobile Application Programming

## ET4710

PGS. TS. Đỗ Trọng Tuấn  
*Viện Điện tử Viễn thông \* Đại học Bách Khoa Hà Nội*

ONE LOVE. ONE FUTURE.