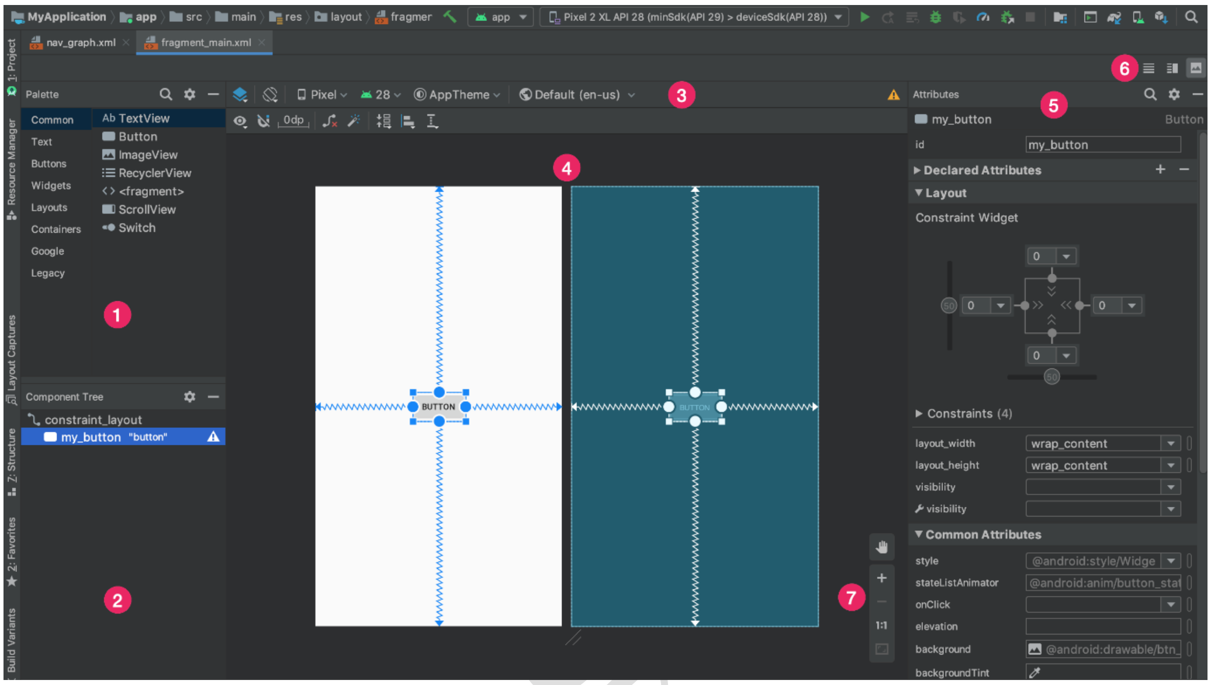
Android Development with Kotlin

Basic Layouts in Android

Layout Editor in Android Studio

The Layout Editor enables one to quickly build layouts by dragging UI elements into a visual design editor instead of writing layout XML by hand. The design editor can be used to preview the designed layout on different Android devices and versions, and can dynamically resize the layout to ensure that it works well on different screen sizes.
The Layout Editor is especially powerful when building a layout with Constraint Layout, a layout manager that's compatible with Android 2.3 (API level 9) and higher.



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.

When you open an XML layout file, the design editor opens by default, as shown in figure 1. To edit the layout XML in the text editor, click the **Code** button in the top-right corner of the window. Note that the **Palette**, **Component Tree**, and **Attributes** windows are not available while editing your layout in **Code** view.
Layout Editor has been extensively covered in the lecture. To explore more about the editor and its functionalities visit the official android developers link:
https://developer.android.com/studio/write/layout-editor

UI Components

In the Android SDK documentation, there are 4 terms used to refer to different parts of the user interface (UI) class hierarchy:

- **View** - Refer to the android.view.View class, which is the base class of all UI classes. android.view.View class is the root of the UI class hierarchy. So from an object point of view, all UI objects are View objects.
- **ViewGroup** - Refer to the android.view.ViewGroup class, which is the base class of some special UI classes that can contain other View objects as children. Since ViewGroup objects are also View objects, multiple ViewGroup objects and View objects can be organized into an object tree to build complex UI structure.
- **Widget** - Refer to any UI class that provides a specific simple or complex UI function. A simple Widget can use a single View object. A complex Widget can use a View object tree with many ViewGroup objects and View objects.
- **Layout** - Refer to any UI class that is specially designed to place child View objects into a UI pattern. Obviously, a Layout has to be a subclass of the android.view.View.ViewGroup.

Examples of UI classes:

- The android.widget.Button class is a Widget class that represents a simple UI component with a single View object.
- The android.widget.LinearLayout class is a Widget class and also a Layout class that represents a complex UI component with a ViewGroup object containing other View objects as children. Child View objects in a LinearLayout object are placed on the UI in a linear pattern.
- The android.widget.DatePicker class is a Widget class that represents a very complex UI component with a ViewGroup object containing year, month, and day spinners. Each spinner is a ViewGroup object containing years, months, or days as View objects for users to select.

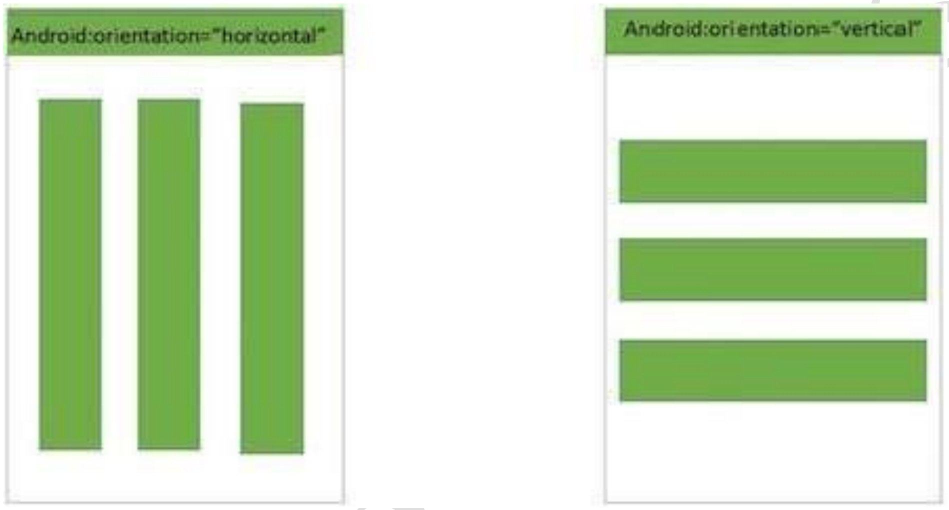
When building a UI, one can search and find a Widget class provided in the Android SDK that match the UI structure. A new class can be extended from a Widget class to add project specific customizations.
One can also build a UI from scratch. This can be achieved with a ViewGroup object, followed by the addition of a child ViewGroup or View objects repeatedly to build a View object tree. Each View or ViewGroup object can then be customized to complete the UI structure.
Some of the most commonly used Android views and widgets are as follows:

- EditText
- ImageView
- TextView
- Button
- ImageButton
- CheckBox

Layout

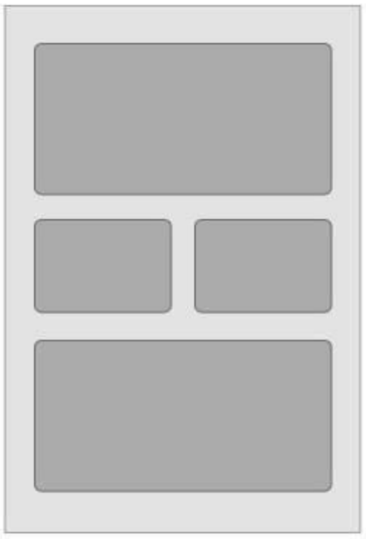
Layouts are subclasses of ViewGroup class and a typical layout defines the visual structure for an Android user interface and can be created either at run time using View/ViewGroup objects or you can declare your layout using simple XML file main_layout.xml which is located in the res/layout folder of your project. There are number of Layouts provided by Android which you will use in almost all the Android applications to provide different view, look and feel. Some of the Android Layouts are discussed below:

- **Linear Layout**
  LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally. It is important to specify the orientation of a linear layout and the height and width of its child objects. The height or width can have its value as zero in vertical and horizontal layouts respectively if weights are used. The weights of an object determine the amount of space the object uses within its parent layout, the higher the weight, the more the area occupied.
  Read more about Linear Layout at
  https://developer.android.com/guide/topics/ui/layout/linear


Android:orientation="horizontal"
Android:orientation="vertical"

Horizontal and Vertical Linear Layouts

- **Relative Layout**
  RelativeLayout is a view group that displays child views in relative positions. The only important values that are to be specified are the heights ad widths of the children as well as the layout itself. If no other attributes are declared the objects are simply stacked above each other in the Layout.
  Read more about Relative Layout at
  https://developer.android.com/guide/topics/ui/layout/relative



Relative Layout

- **Frame Layout**
  FrameLayout is designed to block out an area on the screen to display a single item. Generally, FrameLayout should be used to hold a single child view, because it can be difficult to organize child views in a way that's scalable to different screen sizes without the children overlapping each other.
  Read more about Frame Layout at
  https://developer.android.com/reference/android/widget/FrameLayout

Attributes

Attributes are normally used to define the behaviour of the UI elements within the layout-xml file, or to provide additional information about elements. There are a number of attributes for each UI component but some of the most commonly used Attributes are as follows:
id : These ids are typically assigned in the layout XML files, and are used to find specific views within the view tree.View IDs need not be unique throughout the tree, but it is good practice to ensure that they are at least unique within the part of the tree you are searching.

```
android:id="@+id/my_id"
```

height & width: As the name itself suggests, both of these attributes describes the height and width of the given view. These are necessary attributes for every view in a xml file.

```
android:layout_width="match_parent"       android:layout_height="match
_parent"
```

padding & margin: Padding is the space inside the border, between the border and the actual view's content. Note that padding goes completely around the content.
Margins are the spaces outside the border, between the border and the other elements next to this view.

```
android:padding="10dp"
android:layout_margin="10dp"
```

gravity & layout_gravity: android:gravity sets the gravity of the content of the View its used on. android:layout_gravity sets the gravity of the View or Layout in its parent.

```
android:gravity="center"
android:layout_gravity="bottom"
```

android:maxHeight : An optional argument to supply a maximum height for this view.

```
android:maxHeight="10dp"
```

android:maxWidth : An optional argument to supply a maximum width for this view.

```
android:maxWidth="10dp"
```

android:scaleType : Controls how the image should be resized or moved to match the size of this ImageView.

```
android:scaleType=" centerCrop"
```

android:hint : Hint text to display when the text is empty.

```
android:hint="Text is Empty"
```

android:inputType : The type of data being placed in a text field, used to help an input method decide how to let the user enter text.

```
android:inputType="number"
```

android:textAppearance : Base text color, typeface, size, and style.

```
android:textAppearance="@style/theme"
```

android:textColor : Text color.

```
android:textColor="#000000"
```

android:textSize : Size of the text.

```
android:textSize="16sp"
```

The attributes in Android are not limited to the ones mentioned in the document as every View has its own set of attributes. And a detailed view of those can be found online on the Android's official Developer website. The link to which is given below.
https://developer.android.com/reference/org/xml/sax/Attributes