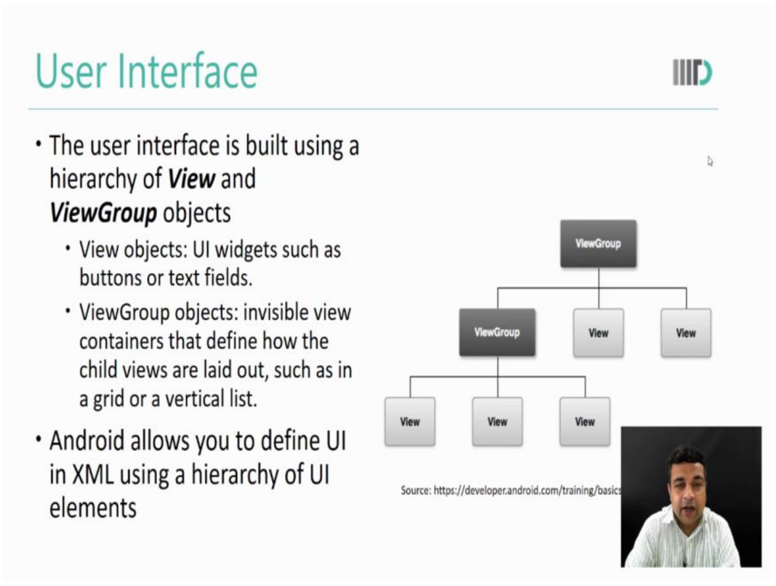
Mobile Computing
Professor Pushpendra Singh
Indraprastha Institute of Information Technology Delhi
Lecture 11
Android Studio User Interface

Hello, now after building two small applications and guiding you to Android Studio and how to create Android Studio projects, let us dig deeply into user interface. As you know that user interface is the most important component in your application and this will be the point at which the user will interact with your application, let us see how Android Studio and Android operating system allows us to create wonderful user interfaces for our applications.

(Refer Slide Time: 0:49)



User Interface

- The user interface is built using a hierarchy of ***View*** and ***ViewGroup*** objects
  - View objects: UI widgets such as buttons or text fields.
  - ViewGroup objects: invisible view containers that define how the child views are laid out, such as in a grid or a vertical list.
- Android allows you to define UI in XML using a hierarchy of UI elements

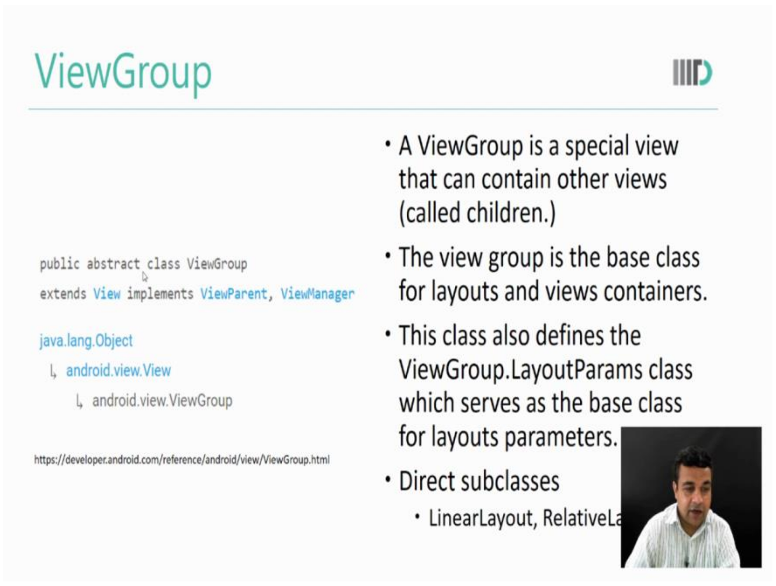
Source: https://developer.android.com/training/basics

A user interface in an Android application is built using a hierarchy of View and ViewGroup objects. The View objects are simple UI widgets such as buttons text fields, radio buttons, etc. While ViewGroup objects are containers, these are the view containers that define, how the child views are laid out, such as in a grid or a vertical list and these are the containers of different other view objects.

If you look at the image, we have a ViewGroup at the top. This ViewGroup has three elements, two View objects and one another ViewGroup and which another, which again has 3 different View objects. This is a sample of a simple user interface. In our application we had in our application we had also created a similar layout we will study that when we open our application.

Android also allows you to define UI in XML using a hierarchy of UI elements as you saw that we could directly manipulate the XML file and add some of these view objects and ViewGroup objects directly into our program without actually writing a code for that. So let us look at the view group.

(Refer Slide Time: 2:23)



ViewGroup

```
public abstract class ViewGroup
extends View implements ViewParent, ViewManager
```

java.lang.Object
  ↳ android.view.View
    ↳ android.view.ViewGroup

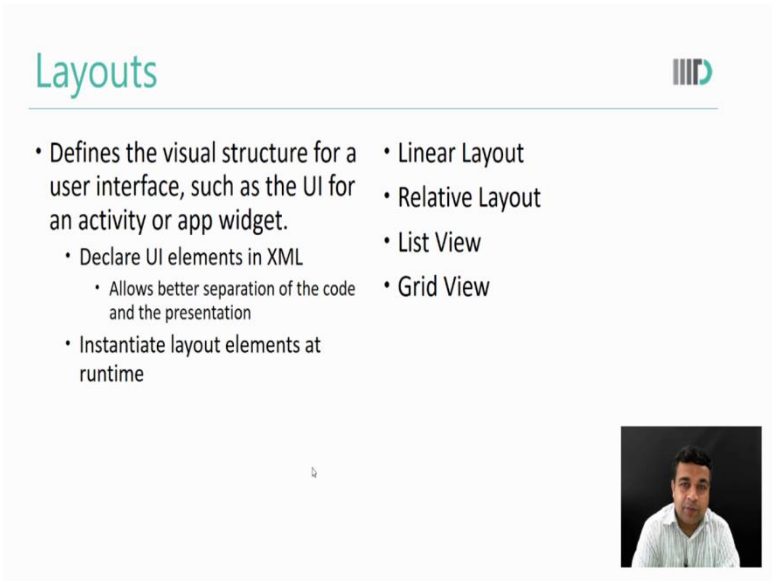
https://developer.android.com/reference/android/view/ViewGroup.html

- A ViewGroup is a special view that can contain other views (called children.)
- The view group is the base class for layouts and views containers.
- This class also defines the ViewGroup.LayoutParams class which serves as the base class for layouts parameters.
- Direct subclasses
  - LinearLayout, RelativeLa

This is a simple definition of view group as given in the Android documentation. As you can see that ViewGroup is an abstract class it extends View and implements two interfaces. This class is defined under Java.lang.object to Andriod.view.View and the andriod.view. ViewGroup.

You can see more details about it, by following this link. So a ViewGroup is a special view that contains other views which are called children. The ViewGroup is the base class for layouts and View contains. In our program we have already used a layout called linear layout. And this class also defines the different ViewGroup Layout parameter which serves as the base class for layout parameters; we will be studying that in details. Some direct sub classes of view group are linear layout, Relative layout and many others. You can get the full information about the subclasses by following this link.

(Refer Slide Time: 3:25)



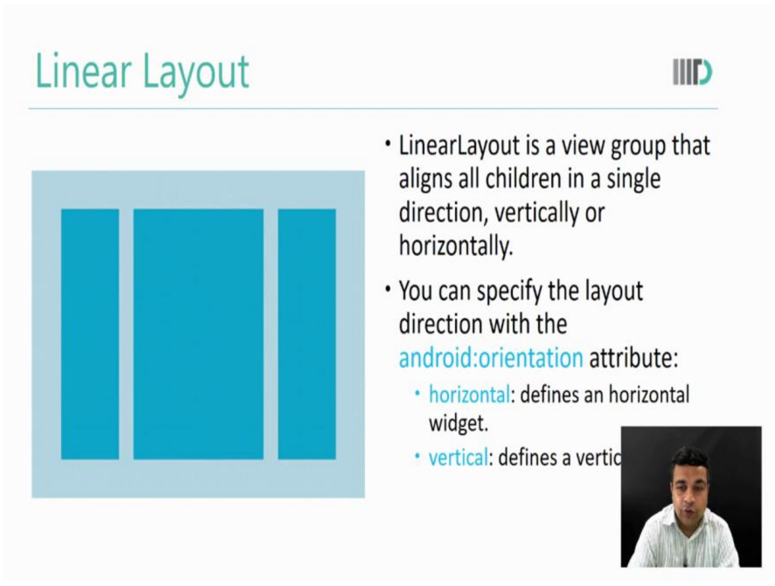
Layouts

- Defines the visual structure for a user interface, such as the UI for an activity or app widget.
  - Declare UI elements in XML
    - Allows better separation of the code and the presentation
  - Instantiate layout elements at runtime
- Linear Layout
- Relative Layout
- List View
- Grid View

Now let us come to the layout when we started our program, our program started with the simple placement of certain view objects. If you remember then in our first application we had a “HelloWorld” string at the top and nothing else and then slowly we removed that “HelloWorld” string and we added another text field. In our second application of (())(3.45) we actually added a text field which was displaying the question and then we added two buttons, now let us see how that is organized inside an android application. So layout defines the visual structure for a user interface such as the UI for an activity or an app widget. We can declare these UI elements in xml or we can instantiate layout elements at one time.

However, declaring these UI elements in XML is a better approach, because it allows the separation of the code and the presentation. You can keep the DOT Java files only for writing logic and.xml file for presentation, so that when we have to make a change in the presentation, we do not have to change the code. Different types of layouts are available in current android app environment; these are mainly linear layout, Relative layout, List View and Grid view.

(Refer Slide Time: 4:54)

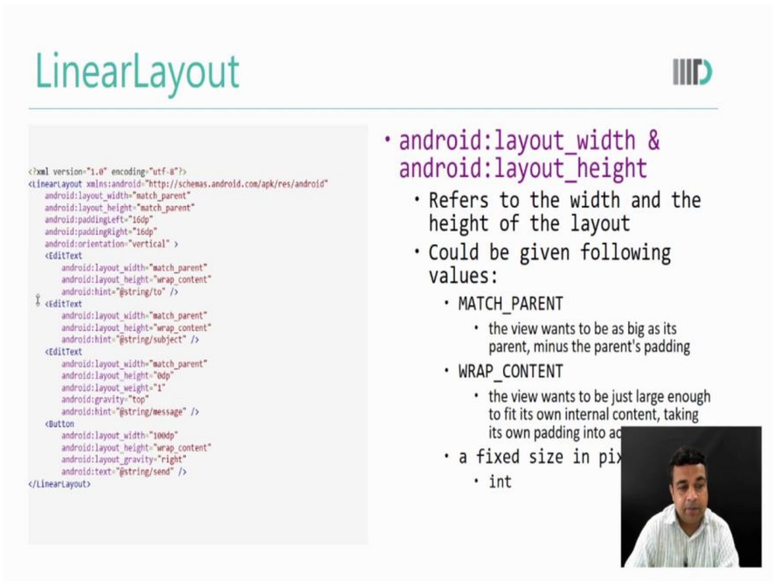


Linear Layout

- 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:
  - horizontal: defines an horizontal widget.
  - vertical: defines a vertic

Let us start with Linear layout, so linear layout is ViewGroup that aligns all children in a single direction either vertically or horizontally. You can specify this direction by using an attribute called Android orientation. As you see in this example, here 1, 2, 3, are defined horizontally similarly, I could have defined them vertically.

(Refer Slide Time: 5:24)



LinearLayout

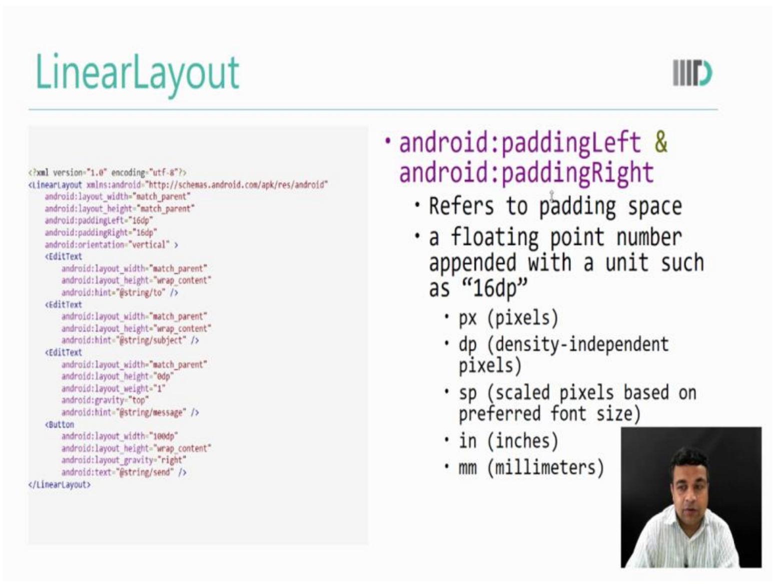
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:orientation="vertical" >
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/to" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/subject" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="top"
        android:hint="@string/message" />
    <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="@string/send" />
</LinearLayout>
```

- android:layout_width & android:layout_height
  - Refers to the width and the height of the layout
  - Could be given following values:
    - MATCH_PARENT
      - the view wants to be as big as its parent, minus the parent's padding
    - WRAP_CONTENT
      - the view wants to be just large enough to fit its own internal content, taking its own padding into ac
    - a fixed size in pix
      - int

Here on the left side we see a sample xml file and on the right side we will try to see each individual element as it appears in the xml file. So let us start with this, as you see the first plan of the xml file gives nothing more than the xml version and then coding (())(5:42) then the second line, the very first line xml ns, which is stands for xml name space android, gives the location of the schema where this, the first line starting with xml ns android gives the location where we can find these xml. If you do not know about xml maybe a good time to read about xml and xml schemas from internet sources.

Now let us start here. The first two elements that you find are android layout grid and android layout height, and both are assigned to a value called Match Parent let us see what these are. So Android layout width and android layout height, it refer to the width and height of the layout (()) (6:35) and these can be given following values. One of the value could be Match Parent, Match Parent means that this view wants to be as big as its parents, minus the padding. Another value could be Wrap Content which means that this view just want to be big enough to contain its internal content and this takes into account the parent. And the third could be that just an integer which gives the fixed size for this in terms of pixels. As you may have guessed Match Parent and Wrap content are also defined as integers.

(Refer Slide Time: 7:18)



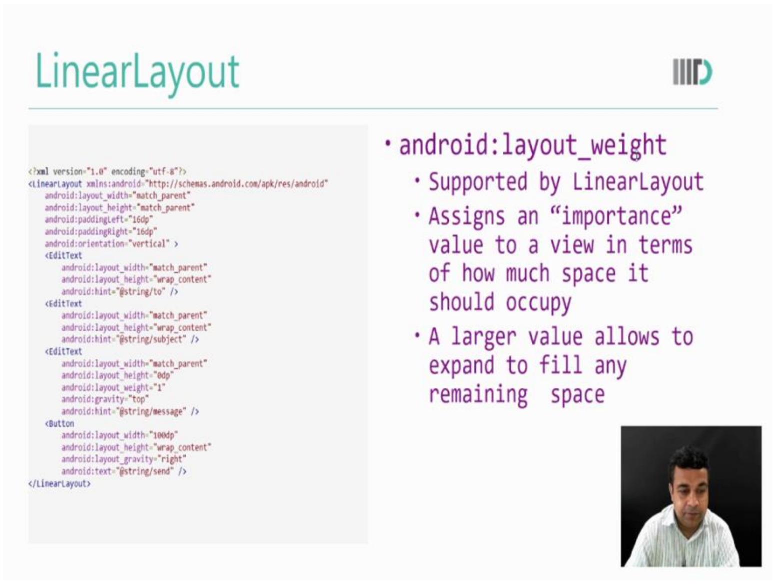
LinearLayout

```
<?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:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:orientation="vertical" >
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/to" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/subject" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="top"
        android:hint="@string/message" />
    <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="@string/send" />
</LinearLayout>
```

- android:paddingLeft & android:paddingRight
  - Refers to padding space
  - a floating point number appended with a unit such as "16dp"
    - px (pixels)
    - dp (density-independent pixels)
    - sp (scaled pixels based on preferred font size)
    - in (inches)
    - mm (millimeters)

Now second element is padding, as you know that padding refers to the space that is available on the sides. So padding left and padding right refers to the to the empty space that is available on the left and the right, so that your element does not look very much touching the boundary. For example if I am displaying a text, I do not want to see that my text is touching the boundaries on the both side and I usually like the view when there is some empty space in between. This empty space is defined by the padding left and padding right. Most often you will see padding left and padding right with something like this 16dp, here the number is a floating point number and it is then appended with a unit. We have different units available to describe the padding size. For example px stands for pixels, dp stands for density independent pixels, we have sp which means scale pixels based on preferred font size, and then we can have inches and millimetres.

(Refer Slide Time: 8:43)



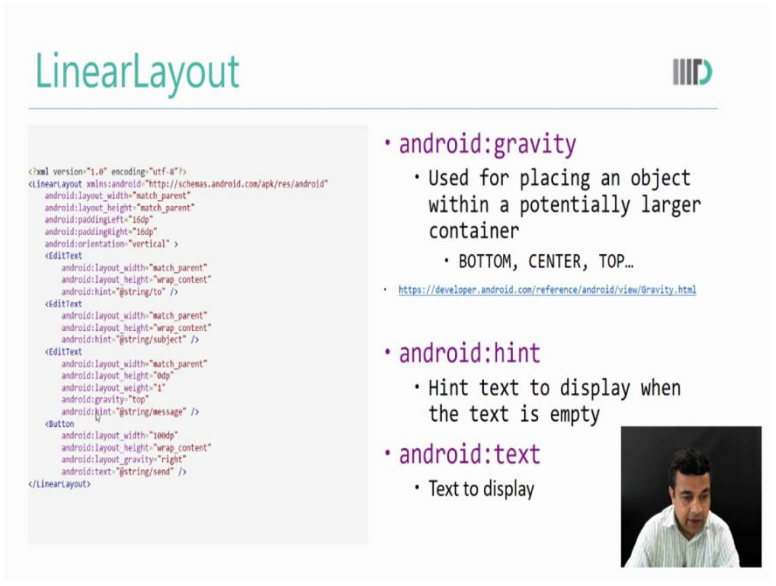
LinearLayout

```
<?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:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:orientation="vertical" >
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/to" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/subject" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="top"
        android:hint="@string/message" />
    <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="@string/send" />
</LinearLayout>
```

- android:layout_weight
  - Supported by LinearLayout
  - Assigns an "importance" value to a view in terms of how much space it should occupy
  - A larger value allows to expand to fill any remaining space

The next element then is the orientation which we have already discussed it could be either vertical or horizontal. Then we may also have something called weight. As you see that in the following xml, we have the weight here. The weight is supported by the Linear Layout and it assigns an "importance" value to a view in terms of how much space it should occupy. So a larger value for weight will allow the particular view object to expand to fill any remaining space. So if I am giving a higher weight to one of the objects, one of the view objects, compared to other view objects, then that particular view objects will then expand to occupy more space.

(Refer Slide Time: 9:26)



LinearLayout

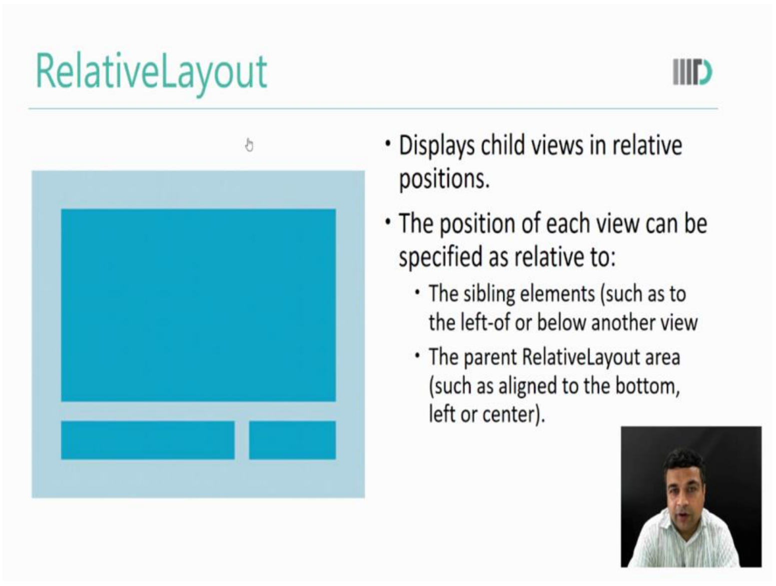
```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:orientation="vertical" >
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/to" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/subject" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="top"
        android:hint="@string/message" />
    <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="@string/send" />
</LinearLayout>
```

- android:gravity
  - Used for placing an object within a potentially larger container
    - BOTTOM, CENTER, TOP…
- https://developer.android.com/reference/android/view/Gravity.html
- android:hint
  - Hint text to display when the text is empty
- android:text
  - Text to display

Gravity let us see, so we have the gravity element here, let us say in top and here the gravity is right, we are defining gravity at two places. So gravity is used for placing an object within potentially larger container. So think of it as a place holder, so if you define it as bottom, then your view object goes to the bottom, if you define it as a centre, your view object goes to the centre, if you define it left it goes to the left, if you define it to top it goes to top and then we have android hint as given here. Hint is simply a hint text which needs to be displayed when the original text just is empty and android text is the text which will be displayed, however if it is empty, then the hint will be displayed.

So now if you go through the xml file again we have described every field that is a field. So far we have not described what Edit text is and what Buttons is, these will be described in a short. So far we have not described what Edit text is and what is Button, we will soon be describing them as well. But as you may have guessed Edit text is a place holder for a text and Button is simply a button as we heard it in our application last time. Then another Layout other than the Linear layout is the Relative layout.

(Refer Slide Time: 11:04)

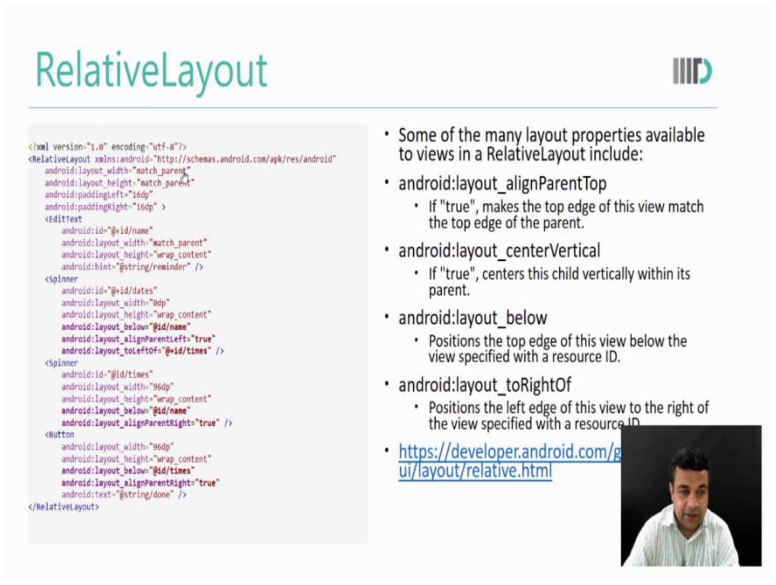


RelativeLayout

- Displays child views in relative positions.
- The position of each view can be specified as relative to:
  - The sibling elements (such as to the left-of or below another view
  - The parent RelativeLayout area (such as aligned to the bottom, left or center).

If you remember when you created your application directly from the Android studio, then the initial xml had the relative layout which we later deleted and we created a Liner layout for our application. What is relative Layout? Relative layout is similar to linear layout; however it displays child view in relative position. So you describe view elements relative to each other. For example, the position of each view will be specified as relative to a sibling elements and relative to the parent element.

(Refer Slide Time: 12:02)



RelativeLayout

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp" >
    <EditText
        android:id="@+id/name"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/reminder" />
    <Spinner
        android:id="@+id/dates"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentLeft="true"
        android:layout_toLeftOf="@+id/times" />
    <Spinner
        android:id="@id/times"
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentRight="true" />
    <Button
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/times"
        android:layout_alignParentRight="true"
        android:text="@string/done" />
</RelativeLayout>
```

- 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.
- https://developer.android.com/g ui/layout/relative.html

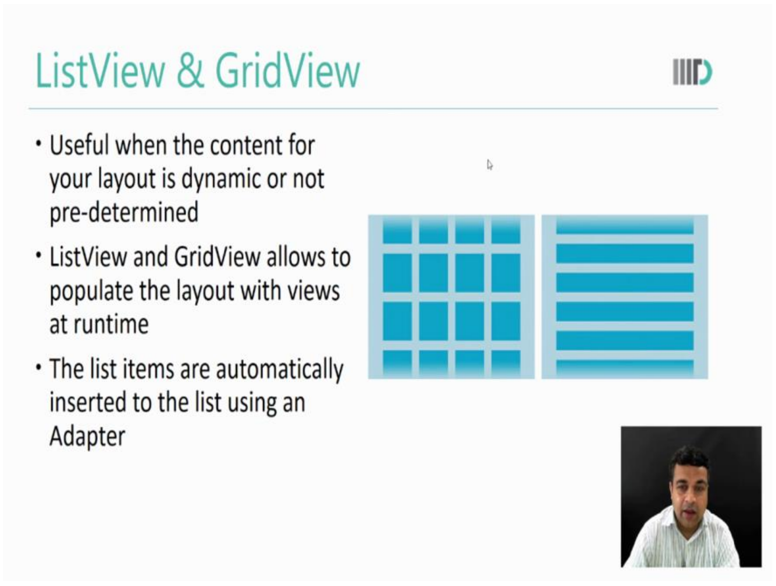
So from parent element you can say oh, it is in the bottom or in the left or in the centre. And then from siblings elements can also be described as the left of the sibling element, or below a sibling element. Let us see a sample xml that uses relative element. Here is the example of xml that is using relative layout. We will not go though each of the page because we have described them. We will only go to the things which are newer than the last slide.

We will only go to the fields which are new compared to the last line. So let us look at something called Andriod:layout_align ParentTop or Align parent left or align parent right. What is means is with respect to the parent if it is true then it will go into the left of the parent, if it is true then it will go to the right of the parent. So for example, layout align parent top, if true makes the top edge of this view match the top edge of the parent.

Similarly if layout is central vertical, if it is true then it centres the child vertically within its parent. If the Layout is below then it positions the top edge, then it position the top edge of this view below the view specified with the resource ID. And if it is layout to Right Of, it positions the left edge of this view to the right of the view specified with the resource ID. Now what is the resource ID that we are referring to? As you see that for this element I have given a resource Id of @id/times and then in the button I am describing it is layout below, below to whom @id/times which means that this button will be below than the (())(13:50) So that is how we define elements in a Relative Layout.

Now you may understand that why we changed the relative layout to the Linear layout and when we started our simple application. Because in our application we were only describing it as field and 2 buttons, which could be more easily described in a Linear layout. Depending on your application requirement you choose either a Liner layout or a Relative layout. More details about relative layout can be find, more details of relative layout can be found on this link.

(Refer Slide Time: 14:34)

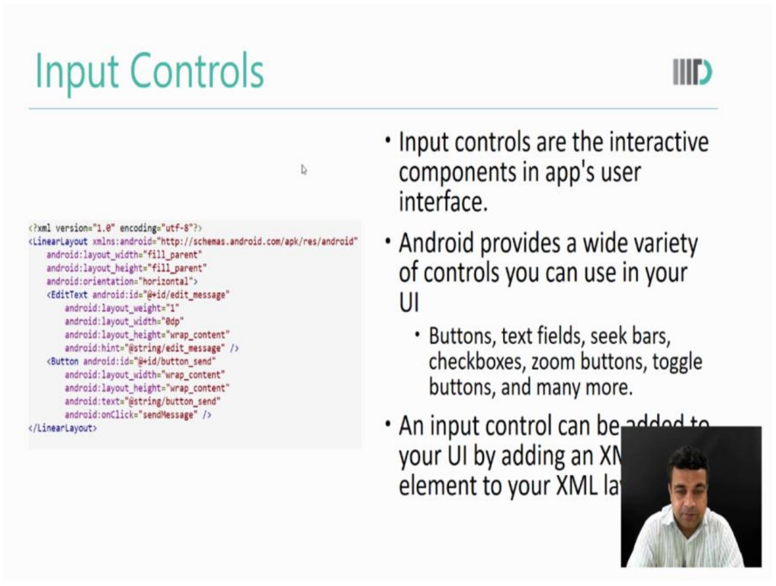


ListView & GridView

- Useful when the content for your layout is dynamic or not pre-determined
- ListView and GridView allows to populate the layout with views at runtime
- The list items are automatically inserted to the list using an Adapter

The other two layouts are list view and grid view these layouts are separate from the previous two layouts and these layouts are useful when the content for your layout is dynamic or it is not predetermined. So list view and grid view layout allows you to populate the layout with views at run time. That is you can add elements into run time. So these list items are automatically inserted to the list using an adapter. We will learn about the adapter later on but as you may imagine, suppose I am going to a list and I want to add a view elements by reading that list, in those cases list view or grid view layout are more suited to my applications.

(Refer Slide Time: 15:24)



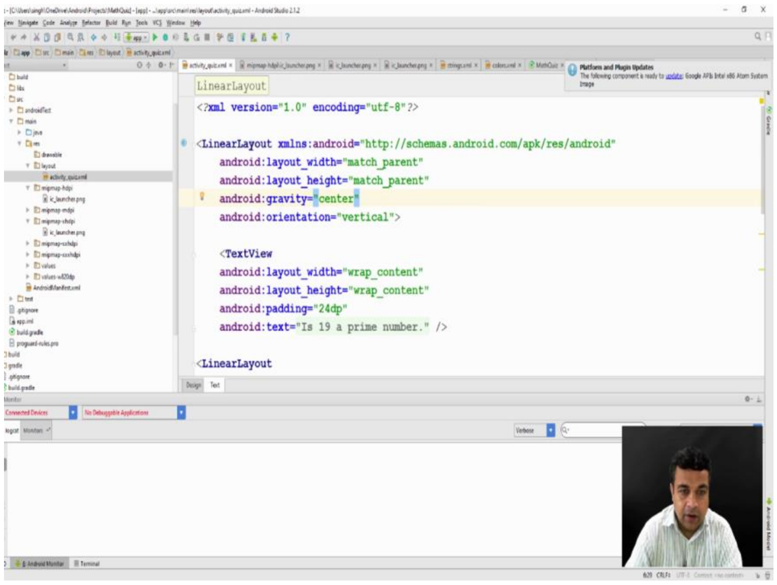
Input Controls

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal">
    <EditText android:id="@+id/edit_message"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:hint="@string/edit_message" />
    <Button android:id="@+id/button_send"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_send"
        android:onClick="sendMessage" />
</LinearLayout>
```

- Input controls are the interactive components in app's user interface.
- Android provides a wide variety of controls you can use in your UI
  - Buttons, text fields, seek bars, checkboxes, zoom buttons, toggle buttons, and many more.
- An input control can be added to your UI by adding an XM element to your XML la

Now after the layout let us understand the input controls in it. Now let us first see the application that we have developed and see how we have defined the layout in those application.

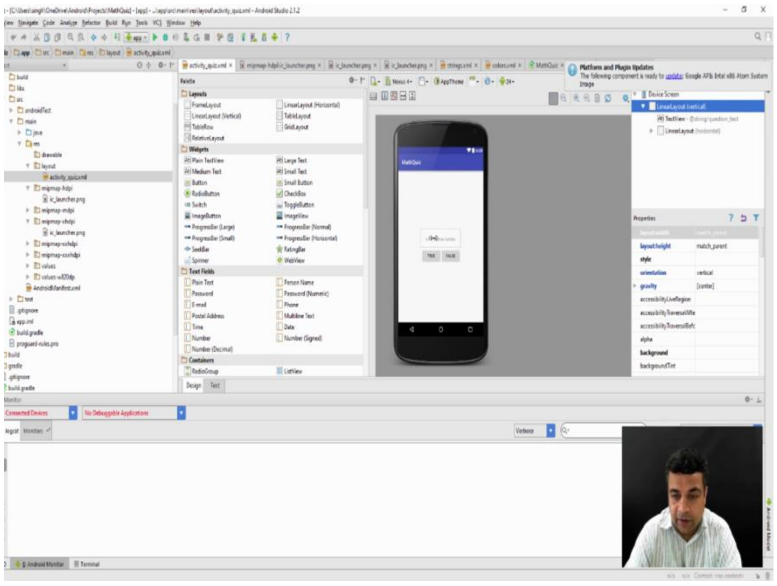
(Refer Slide Time: 15:49)



```
<?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:gravity="center"
    android:orientation="vertical">

    <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="24dp"
    android:text="Is 19 a prime number." />

<LinearLayout
```

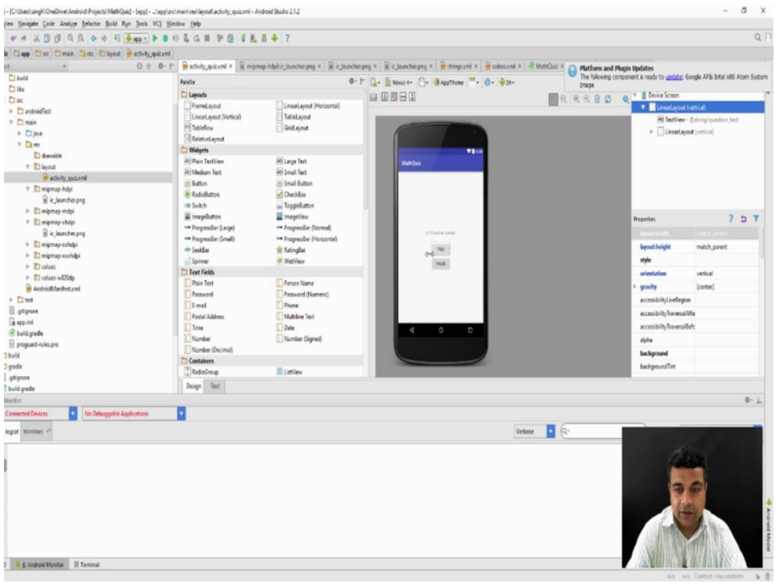
So I am opening my math quiz application, I will go to my activity.quiz.xml file, click and as you can see we are describing a text field and we are describing 2 buttons, so let us go to the text file. So in this we started with the Linear layout, we described the layout, Linear layout to be of the width of match parent and height of the match parent, and the gravity was described as the centre, because the width and height are match parent and the gravity center, with an orientation vertical, the layout will cover the screen completely. Then we described the Text view, where we described it width and height to be Wrap Content that is just enough to have the element inside it, this is string with the padding of 24 dp. Now let us see how it looks in the design.

(Refer Slide Time: 16:54)



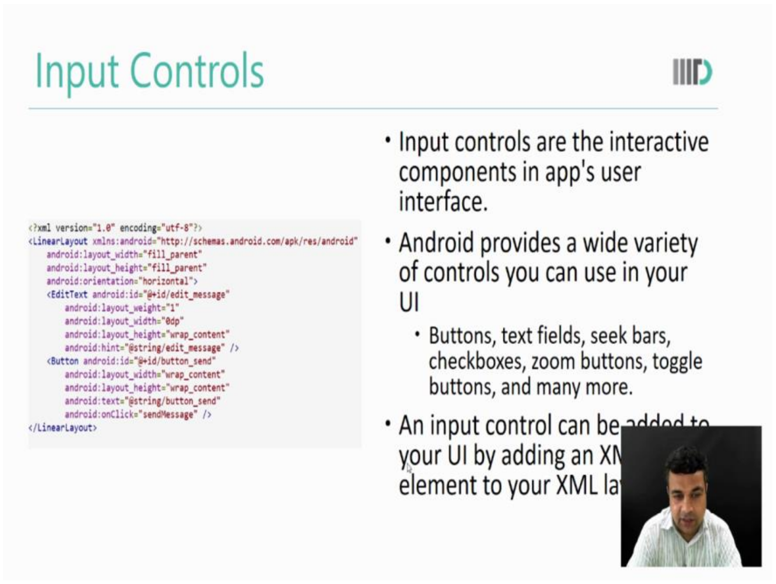
As you can see that this is the box, the box is just big enough to contain the string and then this padding space of 24 dp on each side. After that we describe another Layout, another Linear Layout, and in that linear layout we describe 2 buttons. So essentially we are actually describing another Linear layout here, and in that linear layout we are describing these two buttons. The way we are doing is that, we want that linear layout to be just large enough to contain the objects that are within which means the two buttons and we want them to be in orientation horizontal that is we want to buttons to come from left to right and not from top to bottom. Had we described the layout to be vertical the buttons would change their position, let us see that in action.

(Refer Slide Time: 18:15)



Let me change it to vertical, save it and as you can see that the orientation has changed. Let us change it back. Now for these buttons I am describing an ID and I want to button to be just large enough to contain the content, so it is the true and same thing for another button to just large enough to contain the text fonts and that is what you see here. I end the linear layout inner one and then the outer one. So this is all about the layouts and some of the view objects such as the text view and the button. Now lest go back to our lecture and learn more about input controls.

(Refer Slide Time: 19:16)



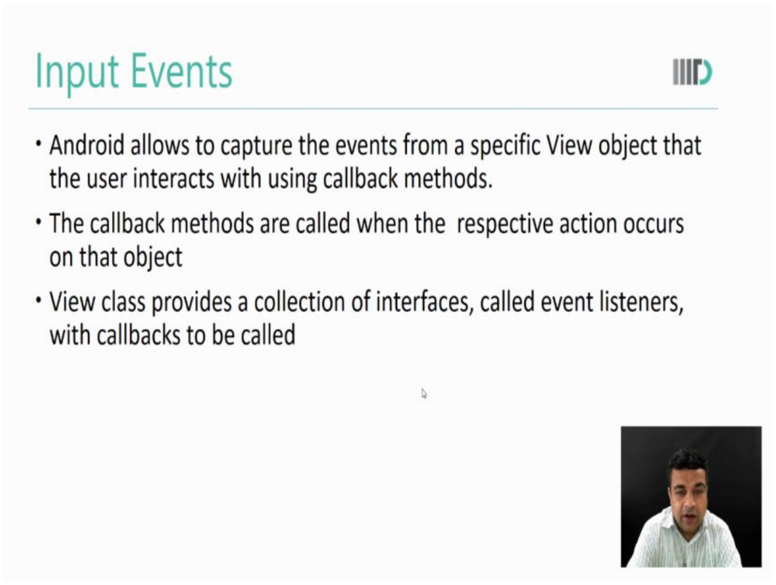
Input Controls

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal">
    <EditText android:id="@+id/edit_message"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:hint="@string/edit_message" />
    <Button android:id="@+id/button_send"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_send"
        android:onClick="sendMessage" />
</LinearLayout>
```

- Input controls are the interactive components in app's user interface.
- Android provides a wide variety of controls you can use in your UI
  - Buttons, text fields, seek bars, checkboxes, zoom buttons, toggle buttons, and many more.
- An input control can be added to your UI by adding an XM element to your XML la

As you saw that in the previous example we were giving our input by pressing the buttons. Similarly, there are other controls available such as text fields seek bars, check boxes, zoom buttons, toggle buttons etc etc. You may, you may know about all of these controls on Android developer documentation, so input controls are the interactive components in your apps user interface. These are the components by which a user interacts with your application and just like the layout you can also add the input control by just modifying the XML file, hence the sample version (())(19.57).

(Refer Slide Time: 20:34)



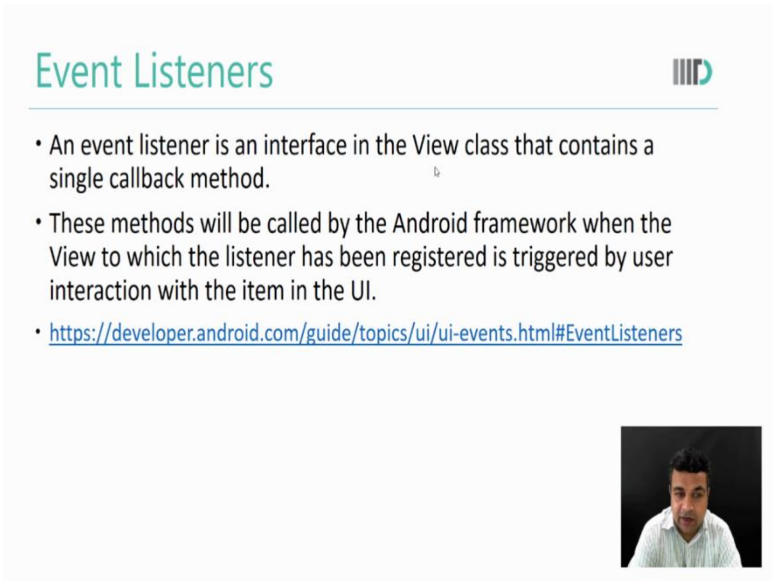
Input Events

- Android allows to capture the events from a specific View object that the user interacts with using callback methods.
- The callback methods are called when the respective action occurs on that object
- View class provides a collection of interfaces, called event listeners, with callbacks to be called

As you can see that we are giving a button here, and the button has an on click enabled, this code means that when this button is pressed, its sends message functions code and on top of this button there is an Edit text, which reads out a message that is defined in the resource edit message. Now android application use what is called event driven program. In event driven programming, usually once you start the program, the program waits for an event to happen. In this case the event could be pressing of a button, or the event could be android application follow what is called event driven programming (())(20:58) as any other UI applications. Event driven programming requires the user to act and based on that action an event is generated, for example when user presses a button an event is generated.

If there is a checkbox and you just select the check box another event is generated and then these events are captured and a callback method is followed. This callback method then defines the process flow. If you do not know about event driven programming, maybe you should read about it separately. Essentially whenever you will do an action a certain method in your program will be called and that method will define your process flow. So for example we have a view object with which a user interacts, the call back method associate it with that view object will be called and will define the process view.

(Refer Slide Time: 22:18)

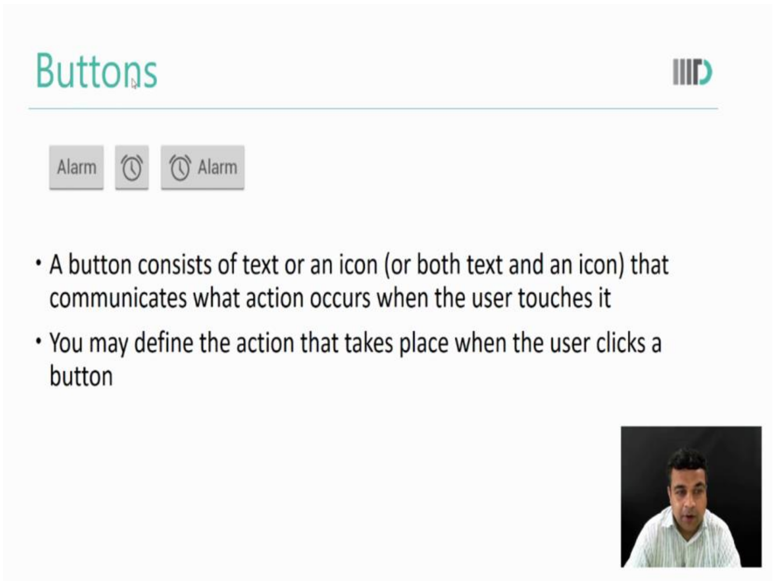


Event Listeners

- An event listener is an interface in the View class that contains a single callback method.
- These methods will be called by the Android framework when the View to which the listener has been registered is triggered by user interaction with the item in the UI.
- https://developer.android.com/guide/topics/ui/ui-events.html#EventListeners

So view class provide a collection of interfaces called event listeners, each of them has its own call back which is then called when that interface is used for the user interaction. So event listeners is an interface in the view class that contains a single call back method, these methods are called by android framework when the view to which the listener has been registered is triggered by user interaction with the item in the UI. If you remember in our math quiz application after we had added the buttons we added the buttons we added an on click listener. That was our way of associating the event listener with a click event of the button. More detail about event listeners can be found by following this link.

(Refer Slide Time: 22:56)



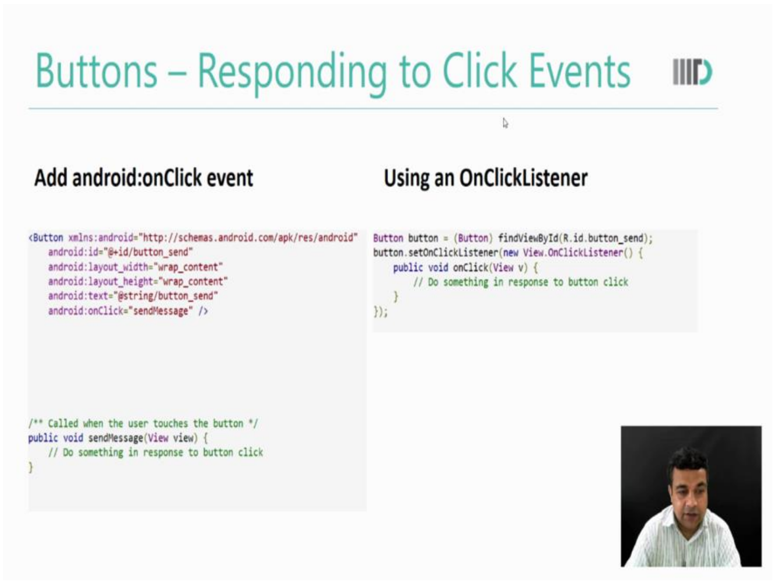
Buttons

Alarm
Alarm

- A button consists of text or an icon (or both text and an icon) that communicates what action occurs when the user touches it
- You may define the action that takes place when the user clicks a button

Now let is look at some of the UI elements that we have used in our programs Buttons; you may have interacted with buttons already because they are so prevalent in all the android applications that have been developed and in any other UI application. In Android, a button consists of text or an icon or both, and that communicates what action occurs when the user touches it. You may define the action that takes place when the user clicks a button.

(Refer Slide Time: 23:26)



Buttons – Responding to Click Events

Add android:onClick event

```
<Button xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/button_send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage" />
```

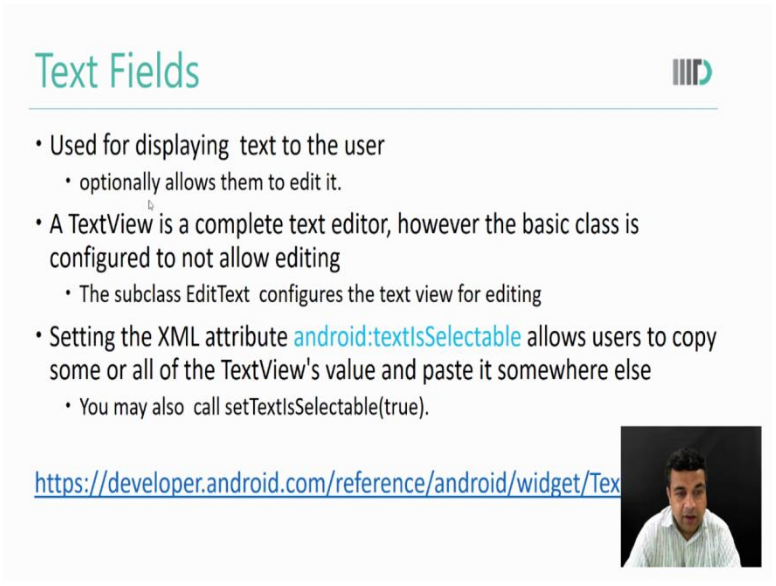
```
/** Called when the user touches the button */
public void sendMessage(View view) {
    // Do something in response to button click
}
```

Using an OnClickListener

```
Button button = (Button) findViewById(R.id.button_send);
button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // Do something in response to button click
    }
});
```

There are two ways in which a button responds to clicks, number one in the XML itself for example, here you can define the message and then later on you can give the complete definition of the message on what happens when the on click event happens. This is one way of registering an on click event with a button. Another way is to use an on click list; this is what we did in our program. So we have a button on that button we attach a listener, that listener then uses the call back method in which we define the functionality that happens when the user clicks that button.

(Refer Slide Time: 24:20)



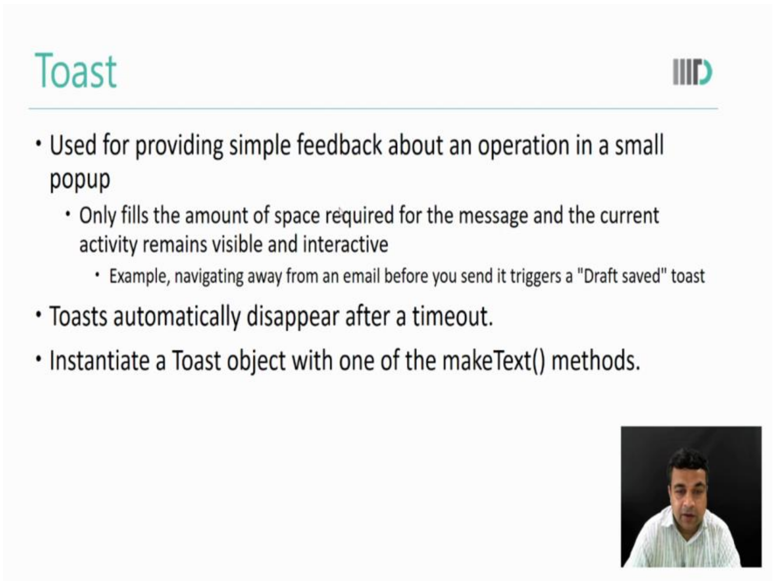
Text Fields

- Used for displaying text to the user
  - optionally allows them to edit it.
- A TextView is a complete text editor, however the basic class is configured to not allow editing
  - The subclass EditText configures the text view for editing
- Setting the XML attribute android:textIsSelectable allows users to copy some or all of the TextView's value and paste it somewhere else
  - You may also call setTextIsSelectable(true).

https://developer.android.com/reference/android/widget/Tex

The other UI elements that we are using in our program is text fields. Text fields are used for displaying test to the user. Optionally you may ask, optionally you may allow the user to edit it. So text field as such is a complete text editor however the basic class is configured to not allow editing. A special sub class called edit text configures the text view for editing. So in our application we added only a text view which was only displaying the message (()) (24:44). However if you want the user to edit it, you may choose instead of text view an edit text as you saw in the XML file of this lecture. Being said the XML attribute andriod:textisSelectable which then allows user to copy some or all of the TextView's value pasted somewhere else. You may also call setTextisSelectable true. More information about textview is available by following the link given here in the slide.

(Refer Slide Time: 25:26)

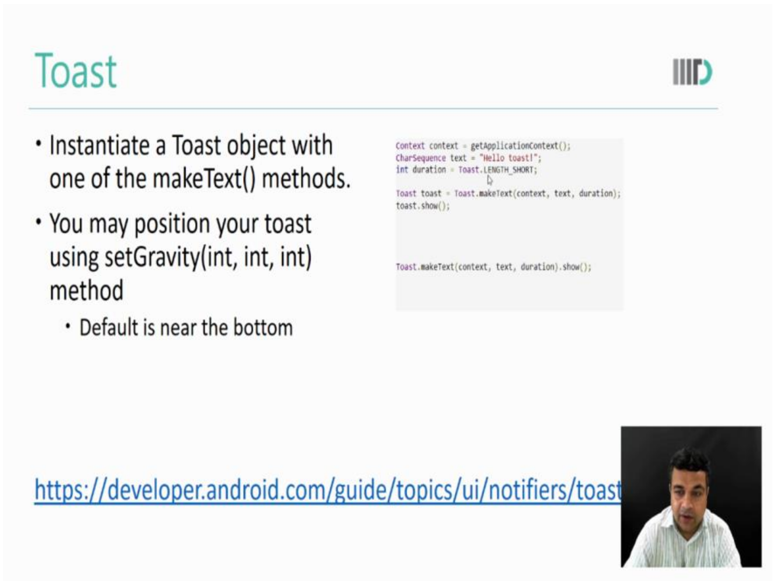


Toast

- Used for providing simple feedback about an operation in a small popup
  - Only fills the amount of space required for the message and the current activity remains visible and interactive
    - Example, navigating away from an email before you send it triggers a "Draft saved" toast
- Toasts automatically disappear after a timeout.
- Instantiate a Toast object with one of the makeText() methods.

The last thing we use in our program is Toast. This was the correct or incorrect message that was poping up when we were pressing true or false button. So Toast are used for providing simple feedback about an operation in a small pop up. Toast only fills the amount of space required for the message and the current activity remains visible and interactive. For example, when you move from a email application normally you get a pop up which says draft saved, that is a Toast for you. We have already created two Toasts in our application when we pressed true or false. Toast also automatically disappear after time out, we usually instantiate a Toast object with one of the make text method, let us look at that in more detail.

(Refer Slide Time: 26:16)



Toast

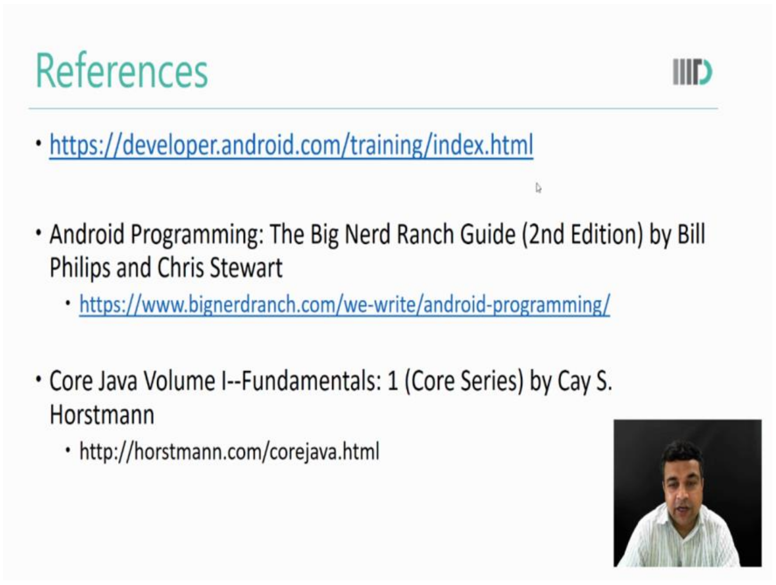
- Instantiate a Toast object with one of the makeText() methods.
- You may position your toast using setGravity(int, int, int) method
  - Default is near the bottom

```
Context context = getApplicationContext();
CharSequence text = "Hello toast!";
int duration = Toast.LENGTH_SHORT;

Toast toast = Toast.makeText(context, text, duration);
toast.show();


Toast.makeText(context, text, duration).show();
```

https://developer.android.com/guide/topics/ui/notifiers/toast

Here is a sample code given. I am creating a make text which takes three parameters one is the context another is the text and the third is the duration and then we do the show. We can directly call the make text from the class because make test is s static method by providing correct parameters. You can instantiate the Toast object with one of the make text methods, there are also other definitions of make text available and you can read about them in the following link. By default your Toast will be near the bottom. If you do not want it near the bottom then you can use the setGravity method to set its position, this is all for today.

(Refer Slide Time: 27:24)



References

- https://developer.android.com/training/index.html
- Android Programming: The Big Nerd Ranch Guide (2nd Edition) by Bill Philips and Chris Stewart
  - https://www.bignerdranch.com/we-write/android-programming/
- Core Java Volume I--Fundamentals: 1 (Core Series) by Cay S. Horstmann
  - http://horstmann.com/corejava.html

We have discussed all the UI elements that we have used in our applications. My advice for you is now go back to the application and while running these slides look at each application component, this will allow you to understand better. For this lecture, I have mainly being using the material available at the android website and I also use the book called the Big Nerd Ranch Guide and previously for Java I have been using the book called Core Java Volume 1, Fundamentals by K S Horstmann. All these books are easily available on (())(27:41).com. Thank you.