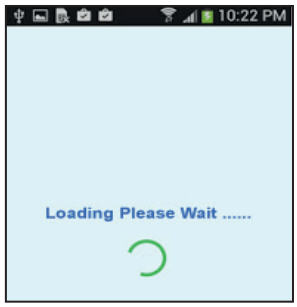
Lesson 8: Android Widgets

Progress Bar (ProgressBar)

The **ProgressBar** is a class used as a user interface element to indicate the progress of an operation. The Progress bar class supports two modes of progress: determinate and indeterminate.

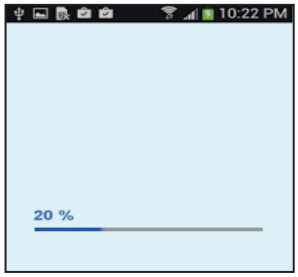
Indeterminate Progress

The Indeterminate mode is the default mode of the progress bar. It is used when you do not know how much time an operation will take. The Indeterminate mode shows a cyclic animation without indicating the amount of progress made.



Determinate Progress (Horizontal)

The determinate mode of the progress bar is used when you want to show how much progress has been made. For example, the percent remaining of a file being retrieved, the amount records in a batch written to database, or the remaining percent of an audio file that is playing.



Example:

The following example explains how you can create and configure a progress bar (Determinate Progress) for a download process**.**

1- Open Android Studio, and then click **File → New → New Project**

2- Select **Empty Activity**, and click **Next**

3- Type: **Lesson08_ProgressBar** for the application name, then click **Finish**.

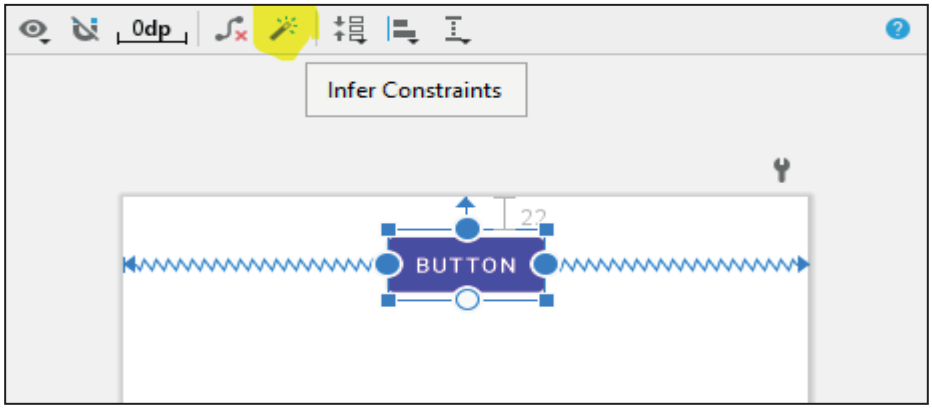
4- Before start with typing the Kotlin code in your app, check the **build.gardle (Module: Lesson08_ProgressBar.app)** file content and be sure it has the Kotlin plugin. If not, add the following code: **id 'kotlin-android-extensions'**

And click the **Sync Now**. The configuration should be as follows:

```
1  plugins {
2      id 'com.android.application'
3      id 'kotlin-android'
4      id 'kotlin-android-extensions'
5  }
```

5- Open the **activity_main.xml** in **Design** mode and **Delete** the "Hello World!" **TextView**.

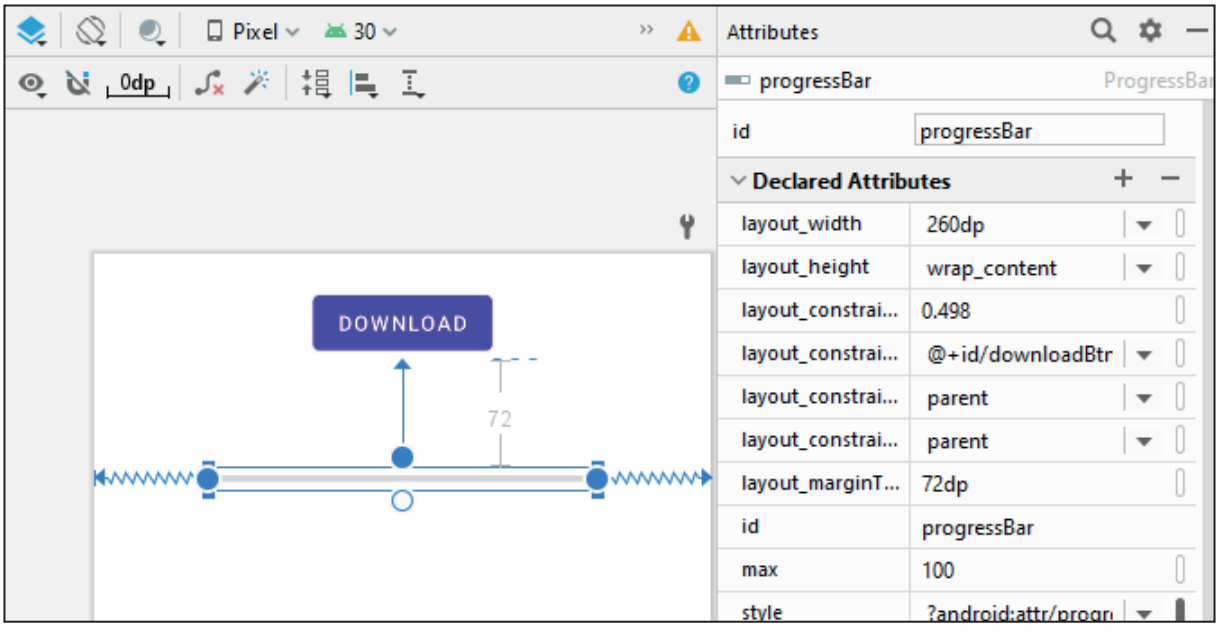
6- Open the **activity_main.xml** file in **Design** mode. Drag and drop a button, set its constraints using the "*Infer Constraints*" tool, and set its **text** attribute value to: **Download**



7- Set the button **Id** attribute value to: **downloadBtn**

8- From the **Palette** panel (Widgets) drag a "**ProgressBar (Horizontal)**" and drop into your activity. Set its constraints and the attribute values of the **ProgressBar** as illustrated in the following figure:

Id: progressBar	layout_width: 260dp	max: 100



Note: The **max** attribute value: 100 means the maximum value of the progress bar is 100.

9- Open the **MainActivity** file and write the code that displays the download process on the progress bar when the user taps the "**Download**" button.

Add a variable "`progressBarStatus =0`" which represents the download process from 0% to 100%, where the starting value is "0".

You also need to add another variable “rate =0” which represents the filling process of the progress bar.

The code as follows:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        var progressBarStatus =0
        var rate =0

    }
}
```

10- Configure the “**downloadBtn**” button to start the download process using progress bar as follows:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        var progressBarStatus =0
        var rate =0
        downloadBtn.setOnClickListener(){}
    }

}
```

11- Add the following code of the while loop which displays the changes in the progress bar every 500 milliseconds.

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        var progressBarStatus =0
        var rate =0
        downloadBtn.setOnClickListener(){ view ->
            Thread(Runnable {

                while (progressBarStatus < 100) {

                    try {
```

```
                        rate += 10
                        Thread.sleep(500)

                    }

                    catch (e: InterruptedException) {
                        e.printStackTrace()

                    }

                    progressBarStatus = rate
                    progressBar.progress = progressBarStatus
                }

            }).start()
        }

    }

}}}
```

Note the following:

`Thread(Runnable {}).start()`: The Thread class will run in the background when you click the download button. This Thread class will also run a Runnable class responsible for displaying its run result on the activity screen. The `start()` method will start the Thread process.

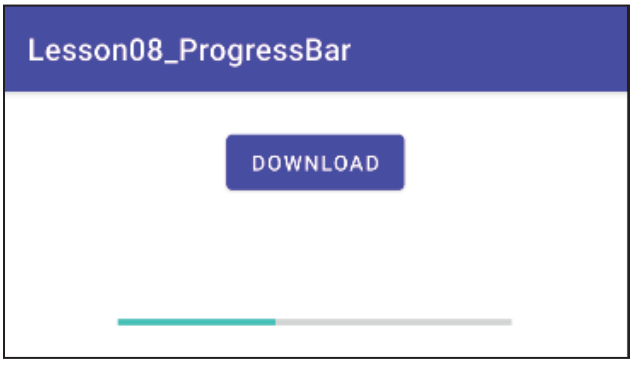
`InterruptedException:` is thrown when a thread is waiting or sleeping and another thread interrupts it using the `interrupt` method in the class `Thread`.

`InterruptedException:` is used here to break out of the while loop.

`Thread.sleep(500):` The Sleep method causes the current thread to suspend execution for a specified period of time (500 milliseconds). This is an efficient way of making the processor time available for the other threads of an application or other apps that might be running on your device.

`e.printStackTrace():` shows the stack trace of the progress bar.

12- **Run** your app, then click the "**DOWNLAOD**" button. You will get the following result:



13- You should hide the progress bar when the download is complete, or when the "**progressBarStatus**" is equal to or greater than 100 by adding the following code:

```
progressBar.setVisibility(ProgressBar.INVISIBLE)
```

The complete code is:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        var progressBarStatus =0
        var rate =0
        downloadBtn.setOnClickListener(){ view ->
            Thread(Runnable {

                while (progressBarStatus < 100) {

                    try {

                        rate += 10
                        Thread.sleep(500)

                    }

                    catch (e: InterruptedException) {
                        e.printStackTrace()

                    }

                    progressBarStatus = rate
                    progressBar.progress = progressBarStatus
                }
                progressBar.setVisibility(ProgressBar.INVISIBLE)

            }).start()
        }}}
```

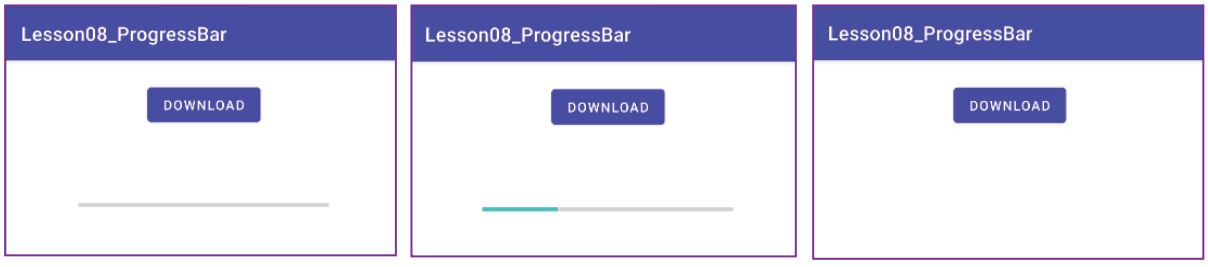
14- Double click the **ProgressBar** class, click the red pop-up lamp, and select **Import**

15- **Stop** your app, then **Run** it again. Tap the **Download** button, then note that the **progressbar** will be hidden when it completes its progress or when its **progressBarStatus** value is: 100.

Before clicking the "DOWNLOAD" button.

After clicking the "DOWNLOAD" button.

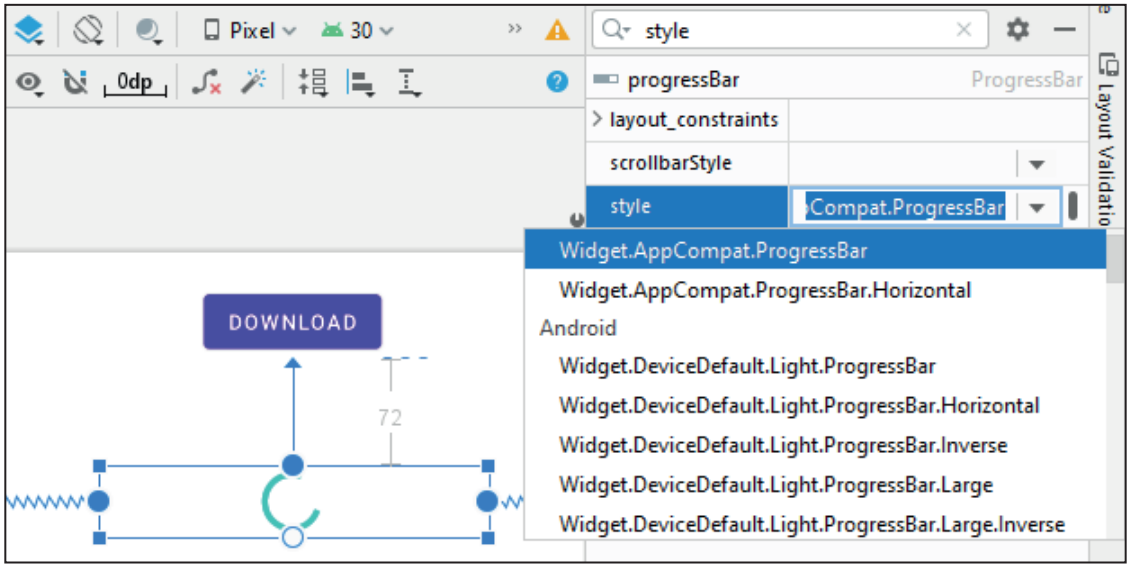
When the progressBarStatus is equal to 100, the progress bar becomes invisible.



16- To change this progress bar to an **indeterminate** progress bar, you can open the **activity_main.xml** in **Code** mode and add the following XML attribute: **android:indeterminate="true"** to the **ProgressBar** tag as illustrated in the following XML code:

```
<ProgressBar
    android:id="@+id/progressBar"
    style="?android:attr/progressBarStyleHorizontal"
    android:layout_width="260dp"
    android:layout_height="wrap_content"
    android:layout_marginTop="72dp"
    android:max="100"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.498"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/downloadBtn"
    android:indeterminate="true"/>
```

17- Change the style attribute value for the progressBar to : **Widget.AppCompat.ProgressBar** as illustrated in the following figure:



18- Stop your app, then Run it again, and note that the progress bar will continue without stopping.

19- The stop of this indeterminate progress bar should be linked with a specific action. This may happen in your app. In this example, we will add a button that will produce this action. To do this, open the **activity_main..xml** file in the **Design** mode, add a button to your activity, set its constraints, change its attribute value for the **text** to **STOP** , change its attribute value for **id** to : **stopBtn** , press Enter (or Return for MAC computer), click **Refactor**.

20- Open the **MainActivity** file and add the following code. This will hide your progress bar when you tap the **Stop** button:

```
stopBtn.setOnClickListener(){
    progressBar.setVisibility(View.GONE)
}
```

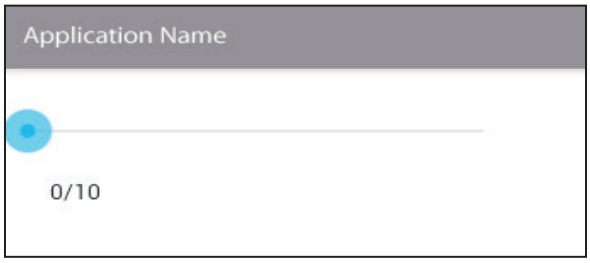
The full code as follows:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        var progressBarStatus =0
        var rate =0
        downloadBtn.setOnClickListener(){ view ->
            Thread(Runnable {
                while (progressBarStatus < 100) {
                    try {
                        rate += 10
                        Thread.sleep(500)
                    }
                    catch (e: InterruptedException) {
                        e.printStackTrace()
                    }

                    progressBarStatus = rate
                    progressBar.progress = progressBarStatus
                }
                progressBar.setVisibility(ProgressBar.INVISIBLE)

            }).start()
        }

        stopBtn.setOnClickListener(){
            progressBar.setVisibility(View.GONE)
        }

    }

}
```

21- **Stop** , then **Run** your app. If you tap the **STOP** button, your progress bar will be hidden.

Seek Bar

A SeekBar is an extension of the ProgressBar with a draggable thumb. The app user can touch the thumb and drag it left or right to set the current progress level or he/she can use the arrow keys. Placing focusable widgets to the left or right of a SeekBar is discouraged.



Example:

The following example explains how you can create and configure a **SeekBar** class.

1- Open Android Studio, and then click **File → New → New Project**

2- Select **Empty Activity**, and click **Next**

3- Type: **Lesson08_SeekBar** for the application name, then click **Finish**.

4- Before start with typing the Kotlin code in your app, check the **build.gardle (Module: Lesson08_ SeekBar.app)** file content and be sure it has the Kotlin plugin. If not, add the following code: **id 'kotlin-android-extensions'**

And click the **Sync Now**. The configuration should be as follows:

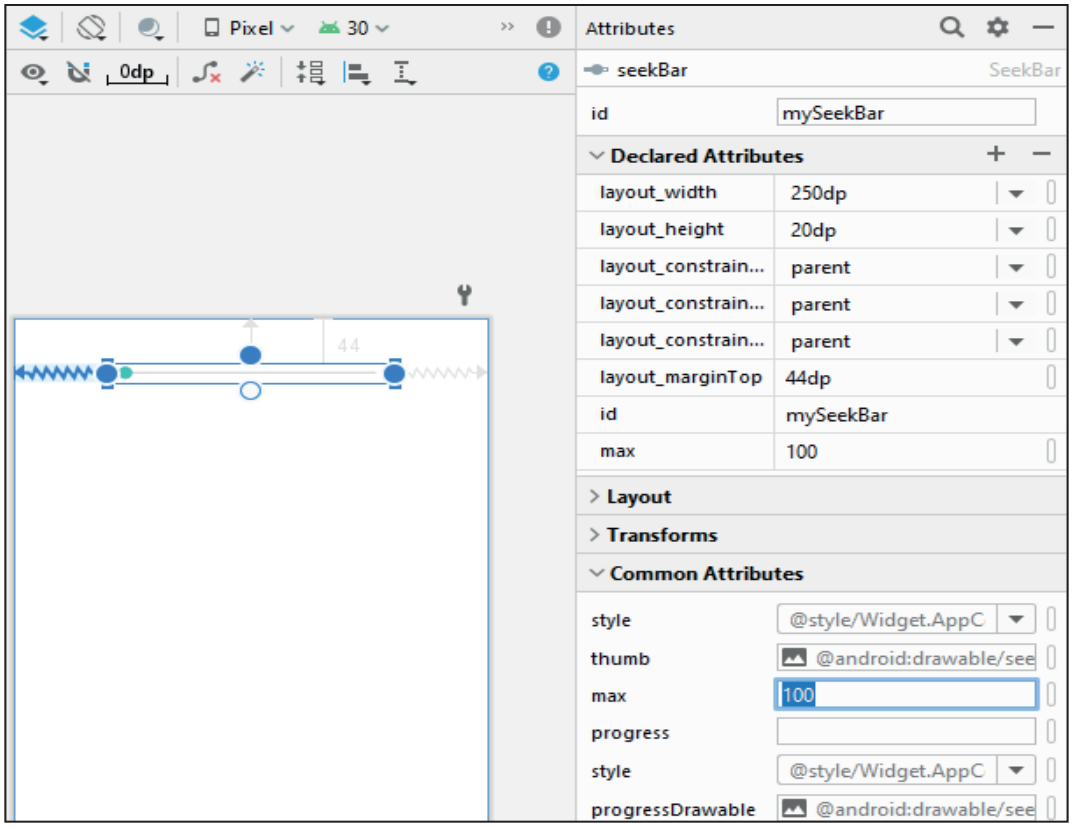
```
1  plugins {
2      id 'com.android.application'
3      id 'kotlin-android'
4      id 'kotlin-android-extensions'
5  }
```

5- Open the **activity_main.xml** in the **Design** mode and **Delete** the "Hello World!" **TextView**.

6- Open the **activity_main.xml** in the **Design** mode, then drag and drop a **SeekBar** widget from the **Palette** panel (Widgets) to your activity.

7- Set the **SeekBar** constraints and configure its attributes values as follows:

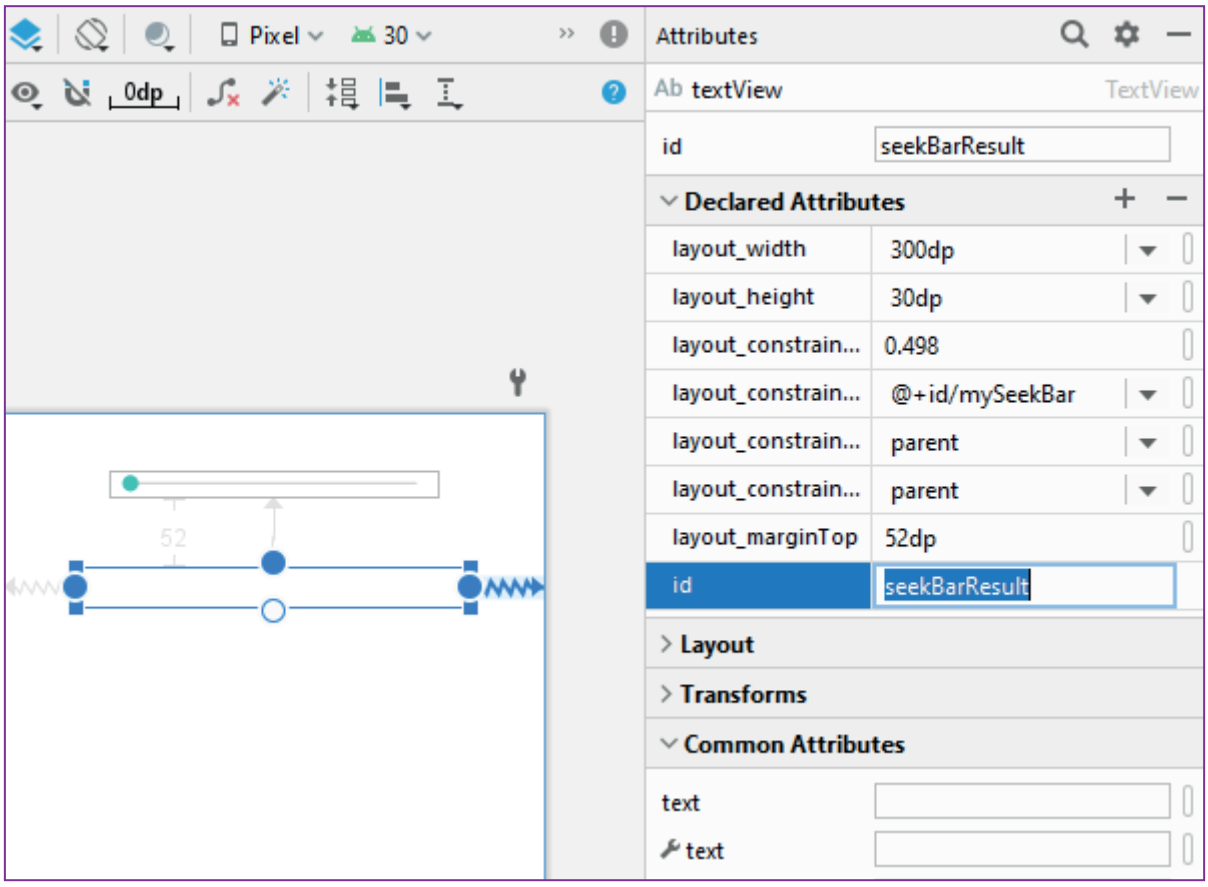
id=**mySeekBar**	layout_width=**250dp**	layout_height=**20dp**	max=**10**



8- Add a **TextView** widget to your activity which will be used later to display the SeekBar values. Set its constraints and change the TextView widget attributes as follows:

ID: **seekBarResult**	textSize: **18sp**	layout_width=**300dp**	layout_height=**30dp**

Set its constraints and delete its **text** attribute value.The **activity_main.xml** in Design mode will be as follows:



9- Open the **MainActivity** file, then write the code which will create a **SeekBar**. You will use two main variables **slider** and **value**, which must be declared as having a non-null data type. These types of variables are called ***Late-Initialized Variables*** and are configured as follows:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        lateinit var slider: SeekBar
        lateinit var value: TextView

    }
}
```

10- In the previous code, double click **SeekBar**, click the red pop-up lamp, and select **Import**. Also, do the same thing for the **TextView** class.

11- In the next step, you will configure the **slider** variable to represent your **SeekBar** using your SeekBar ID and the **value** variable to represent the result which will appear on your **TextView** widget which has the id value : **seekBarResult** as follows:

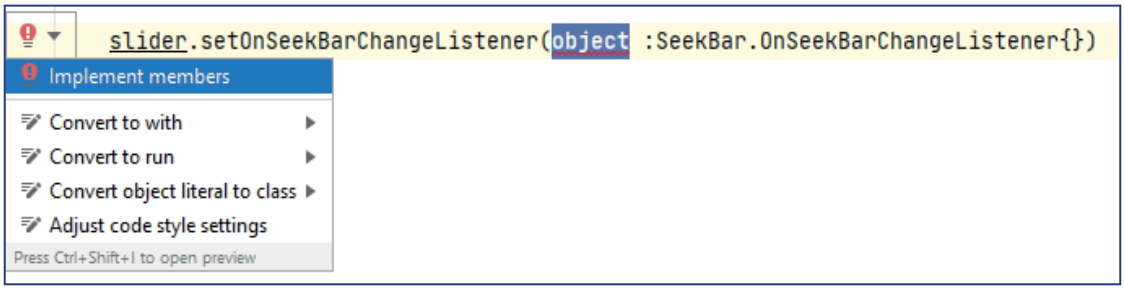
```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        lateinit var slider: SeekBar
        lateinit var value: TextView

        slider=mySeekBar
        value=seekBarResult

    }
}
```

12- In the previous code, double click **mySeekBar**, click the red pop-up lamp, and select **Import**.

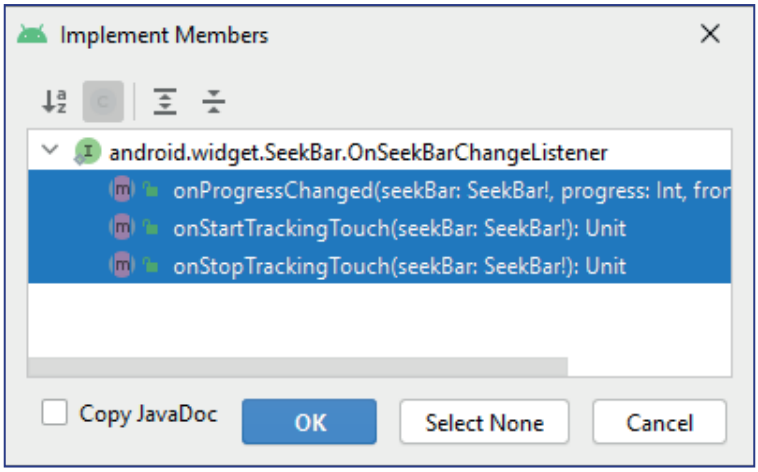
13- Now, you will use the **setOnSeekBarChangeListener** method to configure the SeekBar operations as follows:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        lateinit var slider: SeekBar
        lateinit var value: TextView

        slider=mySeekBar
        value=seekBarResult

        slider.setOnSeekBarChangeListener(object :SeekBar.
OnSeekBarChangeListener{})

    }
}
```

14- As illustrated in the following figure, double click **object**, and click the red pop-up lamp, and select **Implement members**



15- Press **Shift**, select all the three methods as illustrated in the figure below and click **OK**.



These three **SeekBar** methods will be added to your code as illustrated in the following figure:

```
17          slider=mySeekBar
18          value=seekBarResult
19
20          slider.setOnSeekBarChangeListener(object :SeekBar.OnSeekBarChangeListener{
21              override fun onProgressChanged(seekBar: SeekBar?, progress: Int, fromUser: Boolean)
22                  TODO( reason: "Not yet implemented")
23              }
24
25              override fun onStartTrackingTouch(seekBar: SeekBar?) {
26                  TODO( reason: "Not yet implemented")
27              }
28
29              override fun onStopTrackingTouch(seekBar: SeekBar?) {
30                  TODO( reason: "Not yet implemented")
31              }
32          })
33
34
35      }
36  }
```

16- Remove the three **TODO** comments of these **SeekBar** methods. The full code should be as follows:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        lateinit var slider: SeekBar
        lateinit var value: TextView

        slider=mySeekBar
        value=seekBarResult
```

```
slider.setOnSeekBarChangeListener(object: SeekBar.OnSeekBarChangeListener{
            override fun onProgressChanged(seekBar: SeekBar?,
progress: Int, fromUser: Boolean) {

            }

            override fun onStartTrackingTouch(seekBar: SeekBar?) {

            }

            override fun onStopTrackingTouch(seekBar: SeekBar?) {

            }
        })

    }
}
```

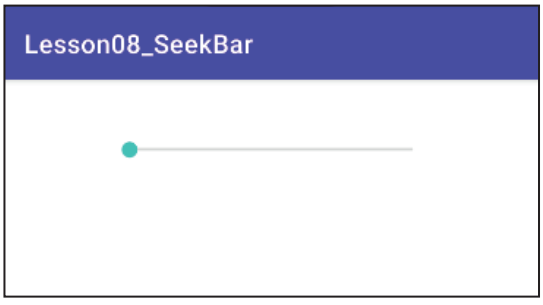
The table below displays the importance and role of each of the three SeekBar methods which have been added to your code in controlling your **SeekBar** behavior:

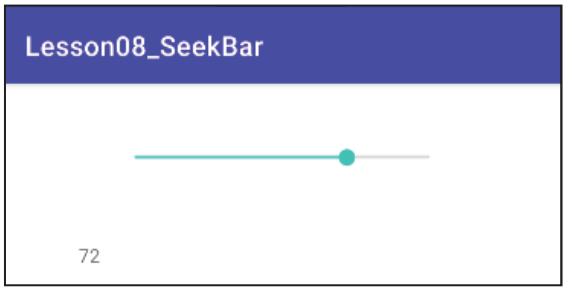
onProgressChanged	Notification that the progress level has changed.
onStartTrackingTouch	Notification that the user has started a touch gesture.
onStopTrackingTouch	Notification that the user has finished a touch gesture.

17- Configure the **onProgressChanged** method to display the progress level change in the **TextView** widget (id: seekBarResult) by adding the following code to your **onProgressChange** method:

```
override fun onProgressChanged(seekBar: SeekBar?, progress: Int,
fromUser: Boolean) {
    value.text=progress.toString()

}
```

18- Run your app and drag the thumb to the right and left side. The run results are as follows:





The values from 0 to 100 (because max attribute value is : 100)

19- Configure the **onStartTrackingTouch** method to display a message only when you press and hold the mouse button at level 0 of the **SeekBar**. When you release the mouse button, the message will hide. This messgae will appear in the **TextView** widget (id: seekBarResult). To do so, add the following code to your **onStartTrackingTouch** method:

```
value.text="Tracking Started ......" + slider.progress
```

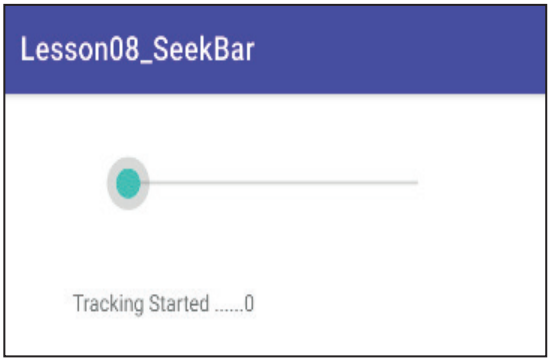
20- Add the code below to your **onStopTrackingTouch** method. This will Configure the **onStopTrackingTouch** method to display a message only when you release your thumb in the TextView widget (id: seekBarResult).

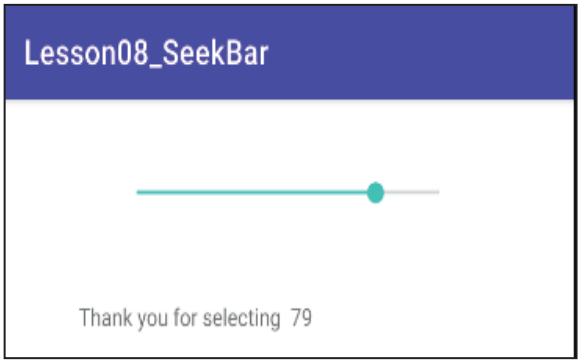
```
value.text="Thank you for selecting  " + slider.progress
```

The full code is:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

 lateinit var slider: SeekBar
 lateinit var value: TextView

 slider=mySeekBar
 value=seekBarResult

slider.setOnSeekBarChangeListener(object :SeekBar.
OnSeekBarChangeListener{

            override fun onProgressChanged(seekBar: SeekBar?,
progress: Int, fromUser: Boolean) {
                value.text=progress.toString()
            }

override fun onStartTrackingTouch(seekBar: SeekBar?) {
   value.text="Tracking Started ......" + slider.progress
            }

override fun onStopTrackingTouch(seekBar: SeekBar?) {
   value.text="Thank you for selecting  " + slider.progress
            }
        })

    }
}
```

21- Stop, Run your app, and the run result as follows:





Date and Time Picker Dialogs

Android provides controls for the user to pick a time or pick a date as ready-to-use dialogs. Each picker provides controls for selecting each part of the time (hour, minute, AM/PM) or date (month, day, year). Using these pickers helps ensure that your users can pick a time or date that is valid, formatted correctly, and adjusted to the user's locale.

Creating a Date Picker:

1- Open Android Studio, and then click **File → New → New Project**

2- Select **Empty Activity**, and click **Next**

3- Type: **Lesson08_Date_Picker** for the application name, then click **Finish**.

4- Before start with typing the Kotlin code in your app, check the **build.gardle (Module: Lesson08_Date_Picker.app)** file content and be sure it has the Kotlin plugin. If not, add the following code: **id 'kotlin-android-extensions'**

And click the **Sync Now**. The configuration should be as follows:

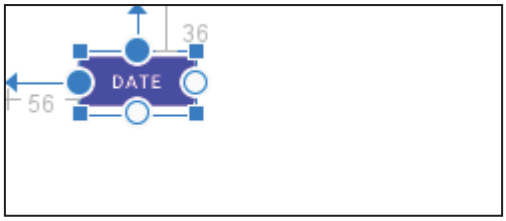
```
1  plugins {
2      id 'com.android.application'
3      id 'kotlin-android'
4      id 'kotlin-android-extensions'
5  }
```

5- Open the **activity_main.xml** in the **Design** mode and **Delete** the "Hello World!" **TextView**.

6- Open the **activity_main.xml** file in the **Design** mode. Drag and drop a button to your activity and set its constraints, then configure its attribute values as follows:

id : **dateBtn**	text : **Date**

Your app interface should be like the following figure:



7- Add a **TextView** widget to the **activity_main.xml** file. Set its constraints, and then configure its attribute values as follows:

id: **dateText**	textSize: **20sp**	layout_width: **wrap_content**	layout_height: **wrap_content**

Delete the **text** attribute value of this **TextView** or set it with a blank value. The activity will look like the figure below:

8- Open the **MainActivity** file and configure the **DATE** button which has the **id**: **dateBtn** to run the date picker code when the app user taps this button using the **`setOnClickListener`** method. The code is:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        dateBtn.setOnClickListener{

        }
    }
}
```

9- Double click the **`dateBtn`**, click the red pop-up lamp, and select **Import**.

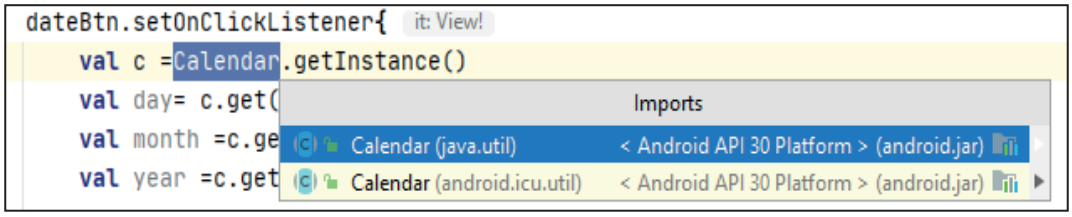
10- When configuring the date in Android, we use the **Calendar** class. This is an abstract class that provides methods for converting between a specific instant in time and a set of calendar fields (methods) such as YEAR, MONTH, DAY_OF_MONTH, HOUR, and so on. It also helps to manipulate the calendar fields, such as getting the dates of the next week.

First, you should declare the variables (components) which your calendar class consists of. These calendar components are : day, month, and year.

The full code is:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        dateBtn.setOnClickListener{
            val c = Calendar.getInstance()
            val day= c.get(Calendar.DAY_OF_MONTH)
            val month =c.get(Calendar.MONTH)
            val year =c.get(Calendar.YEAR)
        }
    }
}
```

11- Double click the **Calendar** class, click the red pop-up lamp, and select **Import**. You should get a sub menu including two choices as illustrated in the following figure. Select: **Calendar(java.util)**



12- In the next step, you are going to configure **DatePickerDialog** class and the constructor of this class as follows:

```
val myDatePicker =
DatePickerDialog(this,android.R.style.ThemeOverlay, DatePickerDialog.On
DateSetListener {DatePicker, Year,Month,Day ->
    dateText.text="$Day/ ${Month+1} /$Year"},year,month,day)

myDatePicker.show()
```

android.R.style.*ThemeOverlay* : is the date calendar theme. You can change it as you like.

DatePicker: the picker associated with the date dialog.

Year: An Integer value that represents the selected year.

Month: An Integer value which represents the selected month (from 0 to 11). Therefore, you should add **"1" as** illustrated in **the** previous code **"**`${Month+1}`**".**

`Day:` An Integer value that represents the selected day of the month (1-31, depending on the month)

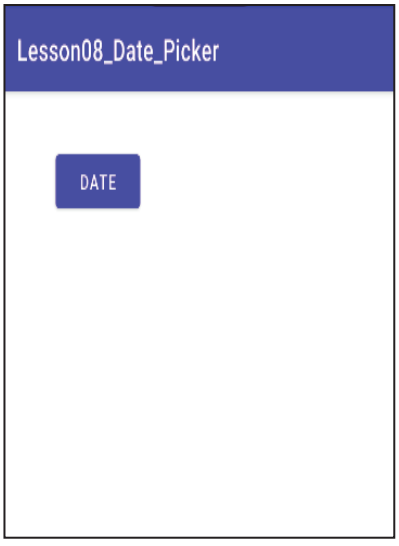
`dateText:` Is the `TextView` widget Id.

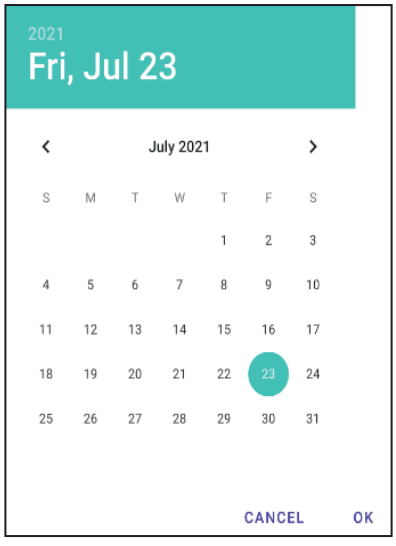
The full code is:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        dateBtn.setOnClickListener{
            val c = Calendar.getInstance()
            val day= c.get(Calendar.DAY_OF_MONTH)
            val month =c.get(Calendar.MONTH)
            val year =c.get(Calendar.YEAR)
            val myDatePicker =
DatePickerDialog(this,android.R.style.
ThemeOverlay,DatePickerDialog.OnDateSetListener {DatePicker,
Year,Month,Day ->
                dateText.text="$Day/ ${Month+1}
/$Year"},year,month,day)
            myDatePicker.show()

        }
    }
}
```

13- Double click the **`DatePickerDialog`**, click the red pop-up lamp, and select **Import**.

14- Run your app, click the **DATE** button, select the date, then click **OK** as illustrated below. The app user will get the date value at the TextView widget (`dateText`).





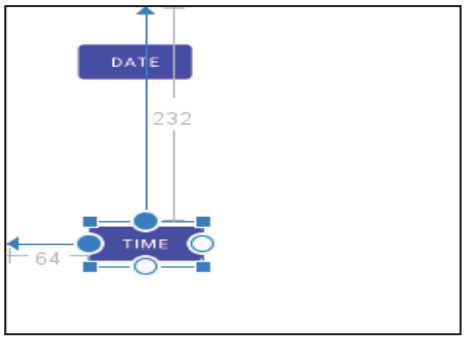
Creating a Time Picker:

Continu using the same **Lesson08_Date_Picker** Android project to add a time picker dialog to your Android app interface by following the steps below:

1- Open the **activity_main.xml** file in the **Design** mode. Add a button to this activity interface using drag and drop technique, set this button constraints, then configure its attributes values as follows:

ID : **timeBtn**	text :**Time**

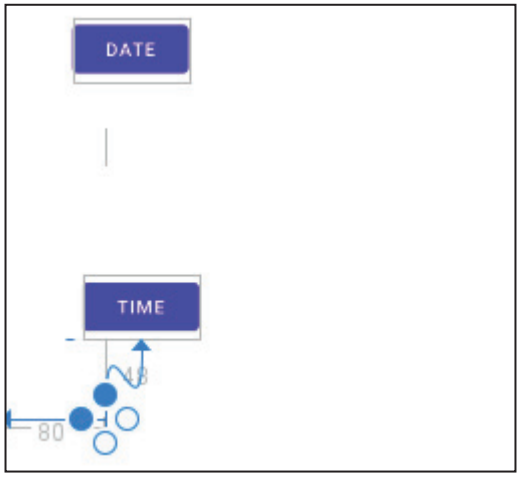
Your activity interface should have the following figure:



2- Add a **TextView** widget to your **activity_main.xml** file. Set its constraints and then configure its attributes values as follows:

Id: **timeText**	textSize: **20sp**	Layout_width: **wrap_content**	Layout_height: **wrap_content**

Delete the **text** attribute value of this "TextView" or set it with a blank value. The activity will look like the figure below:



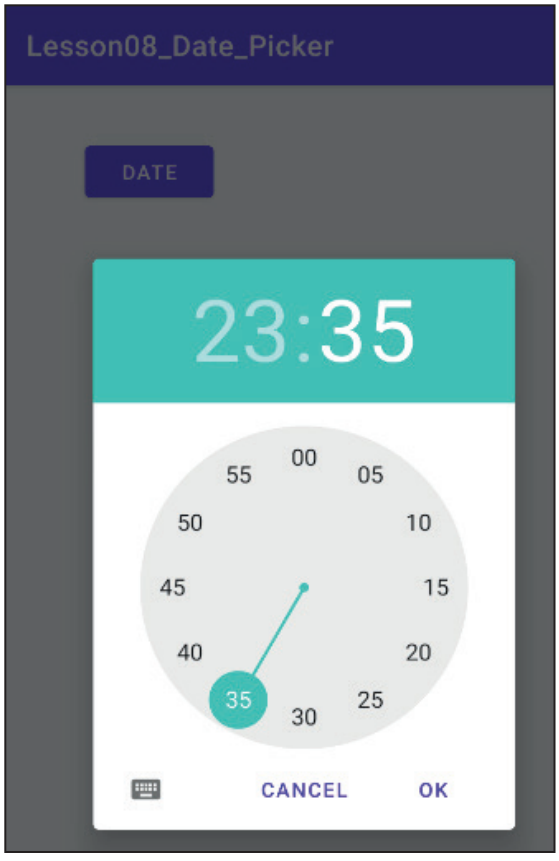
3- Open the **MainActivity** file and add the code below to create a time picker dialog direclty under the previous date picker dialog code:

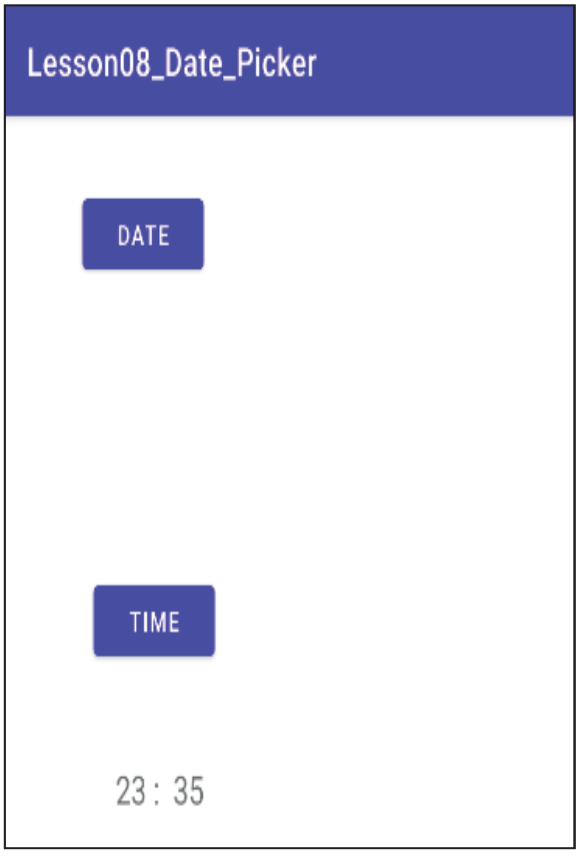
```
timeBtn.setOnClickListener{
    val c = Calendar.getInstance()
    val hour =c.get(Calendar.HOUR_OF_DAY)
    val minutes =c.get(Calendar.MINUTE)

    val myTimePicker= TimePickerDialog(this,TimePickerDialog.
OnTimeSetListener { TimePicker, hourOfDay, minute ->
        timeText.text="$hourOfDay :  $minute"},hour,minutes,true)

    myTimePicker.show()

}
```

4- Double click the **`TimePickerDialog`**, click the red pop-up lamp and select **Import**.

Note: the parameter "**true**" in the code above means that the time format is 24 hours.

5- **Stop** and then **Run** your app. Tap the **TIME** button, select the time, then tap **OK**. (See illustration below.)





Note: When you set the hour, you will be automatically taken to set the minutes. When you click OK, you will get the time as illustrated above. You may also click the keyboard icon to set the time in a digital format.

CalendarView

This widget is used for displaying and selecting dates. The range of dates supported by this calendar is configurable. This widget is similar to **`DatePicker`** widget.

The **`DatePicker`** widget has been in use since API 1; however, the **`CalendarView`** widget was only introduced in API 11. The main difference between the two is the way they look. As they work in much the same way, since the **`DatePicker`** widget is more versatile, this is the widget we normally use.

In this topic, we will introduce how to add a **CalendarView** widget to app interface with some settings briefly.

Example:
1- Open Android Studio, and then click **File → New → New Project**

2- Select **Empty Activity**, and click **Next**

3- Type: **Lesson08_CalendarView** for the application name, then click **Finish**.

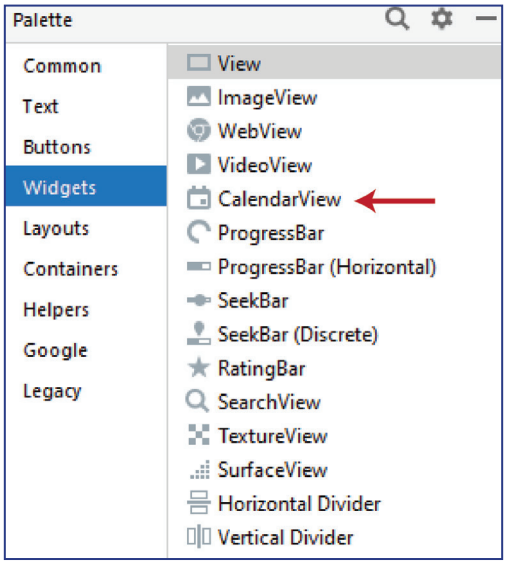
4- Before start with typing the Kotlin code in your app, check the **build.gardle (Module: Lesson08_CalendarView.app)** file content and be sure it has the Kotlin plugin. If not, add the following code: **id 'kotlin-android-extensions'**

And click the **Sync Now**. The configuration should be as follows:

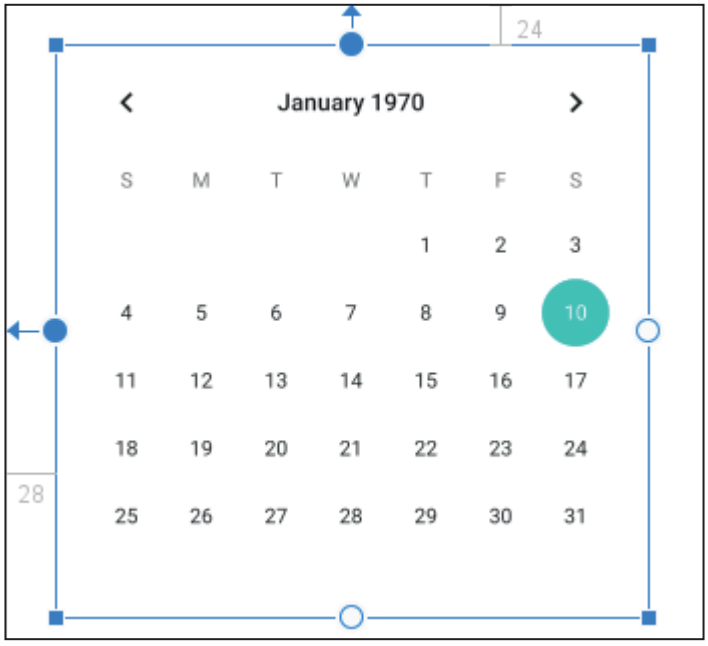
```
plugins {
    id 'com.android.application'
    id 'kotlin-android'
    id 'kotlin-android-extensions'
}
```

5- Open the **activity_main.xml** in the **Design** mode and **Delete** the "Hello World!" **TextView**.

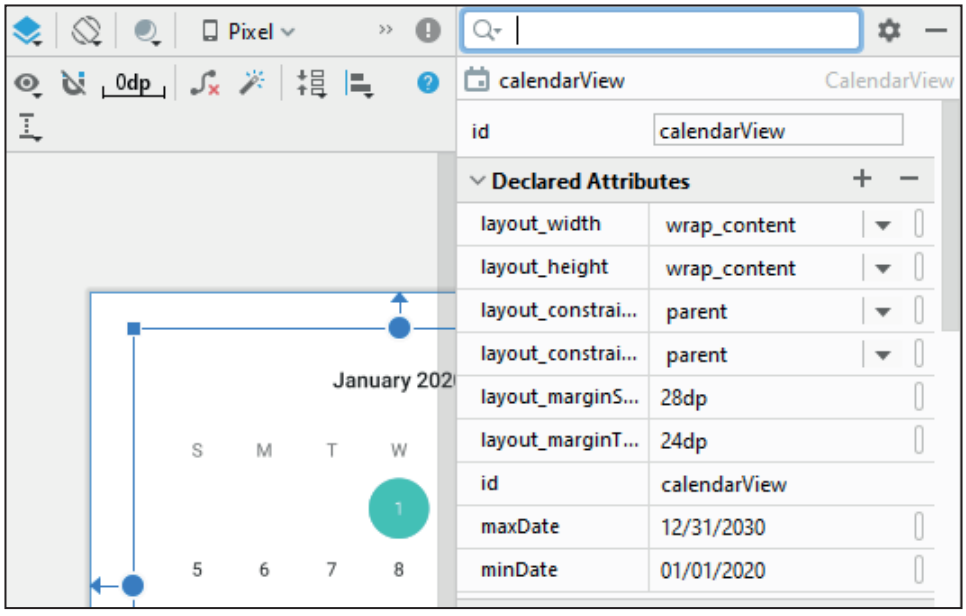
6- From the **Paltte** panle and as illustrated in the figure below, add a **CalendarView** widget to your activity interface using drag and drop technique.



Set your `CalendarView` constraints. You should get interface similar to the following figure:



7- You can set the minimum (From) and maximum (To) date which you app user can only select if you set the **minDate** and **maxDate** attributes values. You can configure them in the XML file or using the **Attributes** panel as illustrated in the following figure:



Note: You may use the search area to find these attributes.

8- **Run** your app and test the date which your app user can select depending on your configurations for the minimum and maximum date. You should get the following :

You may open the MainActivity file, and add a configuration similar to what you did for the DatePicker class to get the date value which your app user may select from your calendar. We will not go into any more details regarding this topic, we just wanted to give you an idea about what choices you have regarding entering a date value to your app.

WebView

You can embed a web browser inside your app activity using a **`WebView`** class or widget. A **`WebView`** widget is a crucial component for many applications that need to display a website without worrying about using Android's native views. For example, an application that displays a list of article titles and loads the full article when clicking on a title can display the article in a web view, instead of retrieving the article's text and doing the rendering and parsing inside the application.

Example:
1- Open Android Studio, and then click **File → New → New Project**

2- Select **Empty Activity**, and click **Next**

3- Type: **Lesson08_WebView** for the application name, then click **Finish**.

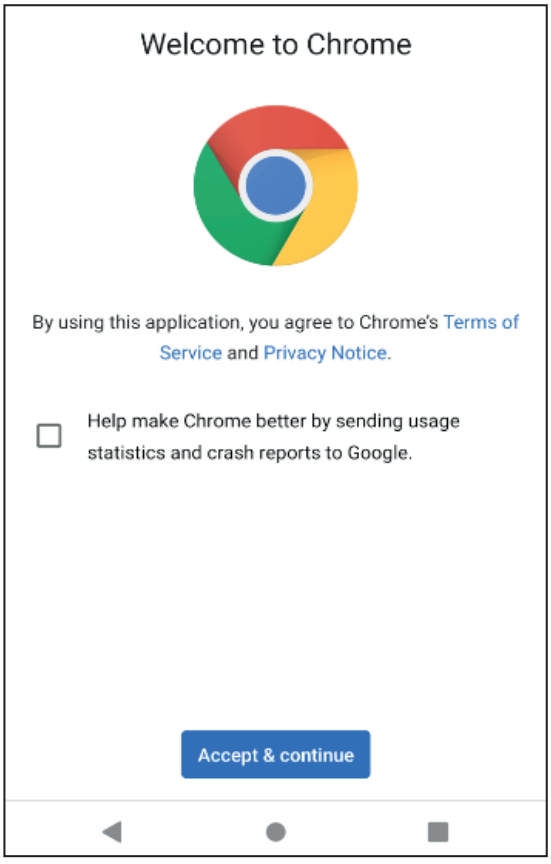
4- Before start with typing the Kotlin code in your app, check the **build.gardle (Module: Lesson08_WebView.app)** file content and be sure it has the Kotlin plugin. If not, add the following code: **id 'kotlin-android-extensions'**

And click the **Sync Now**. The configuration should be as follows:

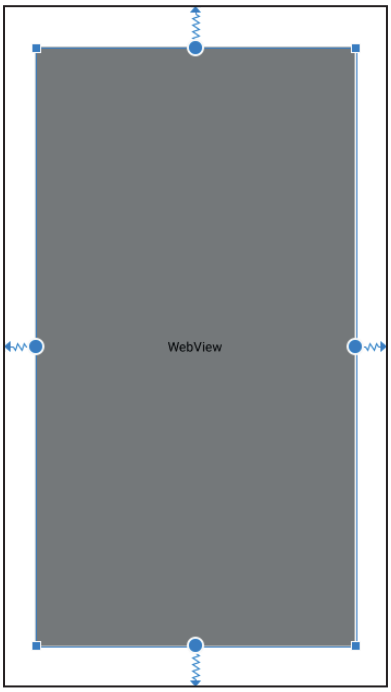
```
plugins {
    id 'com.android.application'
    id 'kotlin-android'
    id 'kotlin-android-extensions'
}
```

5- Open the **activity_main.xml** in the **Design** mode and **Delete** the "Hello World!" **TextView**.

Before starting with the design for the **`WebView`** widget, test your phone emulator web browser and try to browse the web site: https://www.androidatc.com. You should be asked to set the default Chrome web for the first time as illustrated in the figures below. Tap **Accept & Continue.** Then select **No Thanks**. Now, you can be sure to open web sites using this emulator web browser.



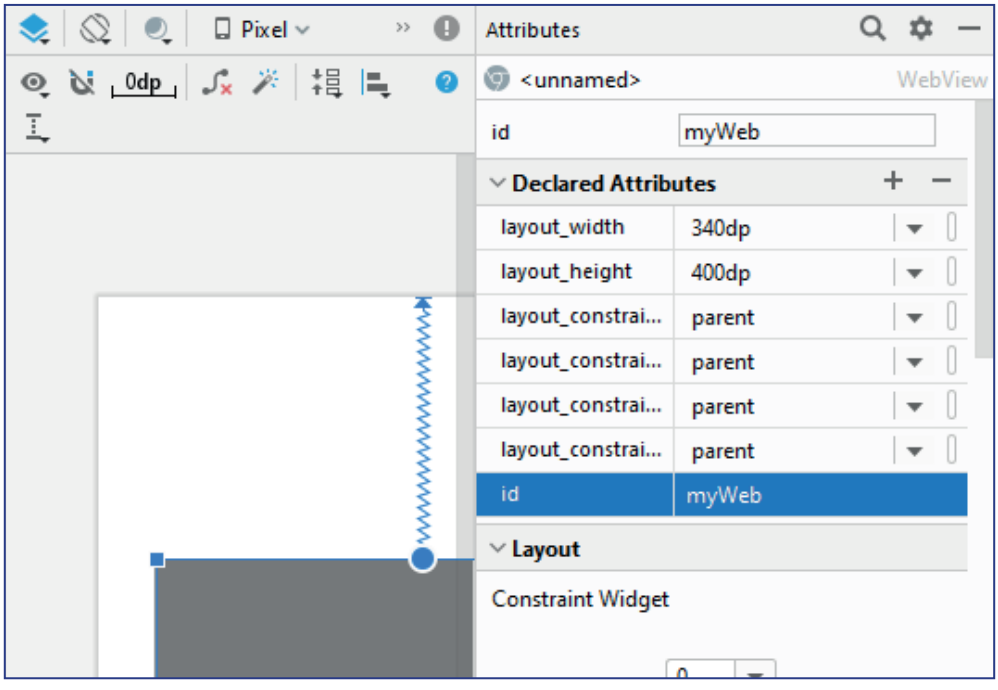
6- Open **activity_main.xml** file in the **Design** mode. From the **Palette** panel (widgets), drag and drop a **WebView** widget to your activity, and set its constraints. Your app activity should have a similar design to the following figure:



7- Set your **`WebView`** widget attributes values as follows:

Id: **myWeb**	layout_width : **340dp**	layout_height: **400dp**

The following figure displays how to set the attributes values of the **`WebView`** widget using the **Attributes** panel.



8- Now, configure your **`WebView`** widget (Id: myWeb) to work as a web browser for the URL: https://www.androidatc.com. To do that, open the **MainActivity** file, and then write the following code:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val myBrowser: WebView = findViewById(R.id.myWeb)
        myBrowser.webViewClient = WebViewClient()

        //Shows the URL
        myBrowser.loadUrl("https://www.androidatc.com")

        //Set the Web View to have a transparent border
        myBrowser.setBackgroundColor(Color.TRANSPARENT)

        //To enable JavaScript for the web browser
        myBrowser.settings.javaScriptEnabled =true

        //to load images automatically
        myBrowser.settings.loadsImagesAutomatically = true

        //Enable Scrolling
        myBrowser.scrollBarStyle= View.SCROLLBARS_INSIDE_OVERLAY

    }
}
```

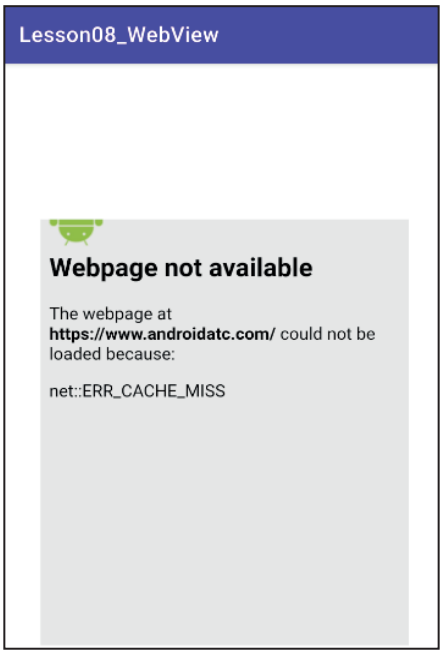
9- Double Click **WebView**, click the red pop-up lamp, and select **Import**

10- Double Click **WebViewClient**, click the red pop-up lamp, and select **Import**

11- Double Click **Color**, click the red pop-up lamp, and select **Import**

12- Double Click **View**, click the red pop-up lamp, and select **Import**

13- **Run** your app. For security reasons all apps don't have permission to connect to the Internet by default; therefore, at this point your app run result is as follows:



14- To give your app the permission to connect to the Internet, configure your app **AndroidManifest** file (app → manifests) to allow your app to connect to the Internet. To do so, write the following gray highlighted XML permission tags before the application tag as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.androidatc.lesson08_webview">
<uses-permission android:name="android.permission.INTERNET" />

<application
  android:allowBackup="true"
  android:icon="@mipmap/ic_launcher"
  android:label="@string/app_name"
  android:roundIcon="@mipmap/ic_launcher_round"
  android:supportsRtl="true"
  android:theme="@style/Theme.Lesson08_WebView">
  <activity android:name=".MainActivity">
     <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
     </intent-filter>
  </activity>

</application>

</manifest>
```

15- **Stop**, and **Run** your app. You should get the following result:

Rating Bar

A RatingBar is an extension of **`SeekBar`** and **`ProgressBar`** that shows a rating in stars. The user can touch/drag or use arrow keys to set the rating when using the default size RatingBar.

The number of stars set (via **`setNumStars(int)`** or in an XML layout) will be shown when the layout width is set to wrap content (if another layout width is set, the results may be unpredictable).

This widget is used to get the rating from the user. The Rating returns a floating-point number. It may be 2.0, 3.5, 4.0 etc.

Android **`RatingBar`** displays the rating in stars. Android **`RatingBar`** is the subclass of **`AbsSeekBar`** class.

The **`getRating()`** method of android RatingBar class returns the rating number.

Example:

In this example, you will create a small Android app consists of one app interface (activity). This interface includes a **RatingBar** widget, **Button**, and **TextView** widgets. When your app user selects his/her rate value (number of stars), then taps the button (Submit button), the rate value will be displayed in the **TextView** widget.

To create this app, follow the following steps:

1- Open Android Studio, and then click **File → New → New Project**

2- Select **Empty Activity**, and click **Next**

3- Type: **Lesson08_RatingBar** for the application name, then click **Finish**.

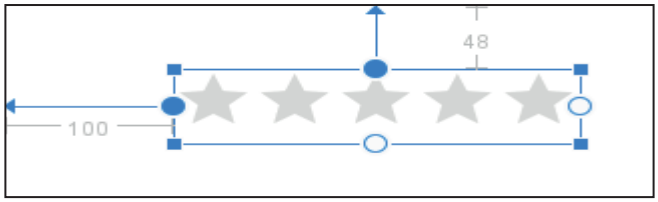
4- Before start with typing the Kotlin code in your app, check the **build.gardle (Module: Lesson08_RatingBar.app)** file content and be sure it has the Kotlin plugin. If not, add the following code: **id 'kotlin-android-extensions'**

And click the **Sync Now**. The configuration should be as follows:

```
plugins {
    id 'com.android.application'
    id 'kotlin-android'
    id 'kotlin-android-extensions'
}
```

5- Open the **activity_main.xml** in the **Design** mode and **Delete** the "Hello World!" **TextView**.

6- From the **Palette** panel (Widgets), add the **RatingBar** widget to your activity using drag and drop technique, and set its constraints. The rating bar should be as illustrated in the figure below:



7- Set the attribute value of your RatingBar ID to: **myRatingBar** and **numStars**: 5

8- Using drag and drop technique, add a **Button** widget to your activity under the RatingBar widget, and set its constraints. Set this button widget attributes values as follows:

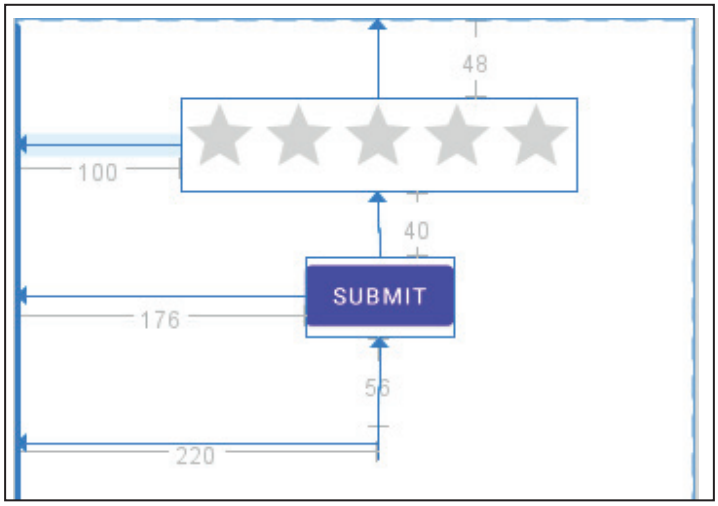
id: **submitBtn**	text: **Submit**

9- Add a **TextView** widget under the **Submit** button and set its constraints. You will use this TextView widget later to display the rating value which will be selected by the app user. Set the attributes values of the **TextView** widget as follows:

id: **rateText**	layout_width: **wrap_content**	layout_height: **wrap_content**

Also, delete the **text** attribute value.

Your activity should have the following figure:



10- Open the **MainActivity** file and configure your **Submit** button (id: submitBtn) to display the rating value using the **`getRating()`** method in the **TextView** widget (id: rateText)

Add the following code:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        submitBtn.setOnClickListener {

        }
    }
}
```

11- Double click the **`submitBtn`**, click the red pop-up lamp, and select **Import**

12- Add the following code to your **setOnClickListener** method:

```
rateText.text= myRatingBar.rating.toString()
```

The **rating** method here represents the **`getRating()`** method, and **`myRatingBar`** is the **id** of your **RatingBar** widget.

The full code as follows:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        submitBtn.setOnClickListener {
            rateText.text= myRatingBar.rating.toString()

        }
    }
}
```

13- **Run** your app, select your rate, and tap the **Submit** button. As illustrated in the following figure, you will get the rate value at your **TextView** (id: **rateText).**

For sure you can configure your app to send this rate value to your cloud Firebase database instead of displaying it on your app interface. This will be clearer when you finish lesson 10 of this course which discusses the Firebase database topic in details.

VideoView

The purpose of using this widget is to display a video file. This widget or class provides a high level view for media playback that can be integrated with either a *SessionPlayer* or a *MediaController* class.

Developers can easily implement a video rendering application using this **`VideoView`** class. By default, a **`MediaControlView`** class which provides the playback control buttons such as play, pause and other control video view buttons is attached on top of **`VideoView`.**

Example:
In this example your will create a simple Android app display a video file.

To create an Android app using this **`VideoView`** widget, follow the following steps:

1- Open Android Studio, and then click **File → New → New Project**

2- Select **Empty Activity**, and click **Next**

3- Type: **Lesson08_ VideoView** for the application name, then click **Finish**.

4- Before start with typing the Kotlin code in your app, check the **build.gardle (Module: Lesson08_ VideoView.app)** file content and be sure it has the Kotlin plugin. If not, add the following code:

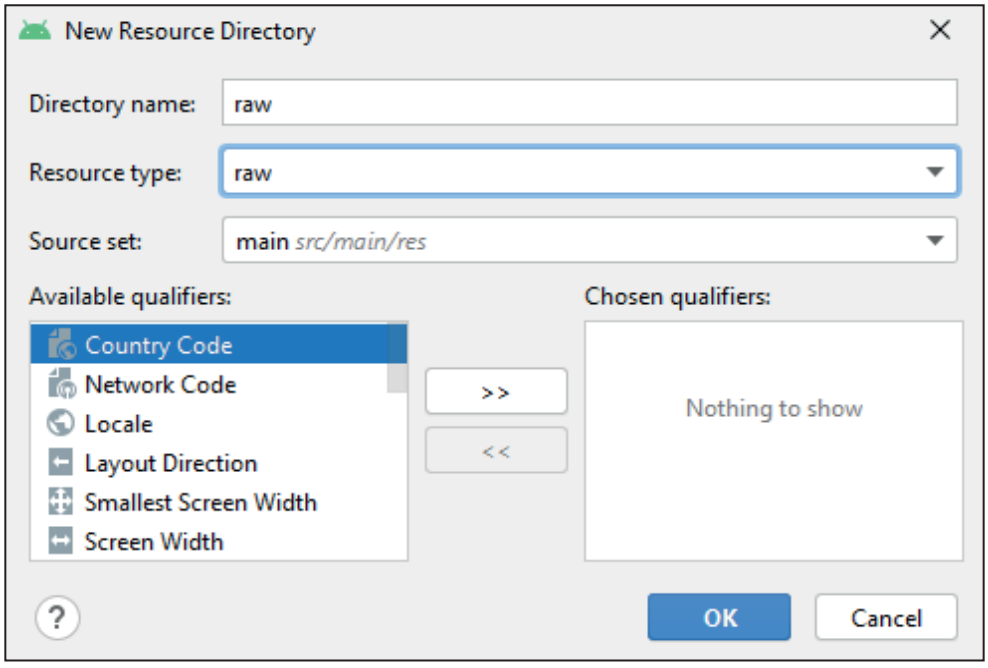
id 'kotlin-android-extensions'

And click the **Sync Now**. The configuration should be as follows:

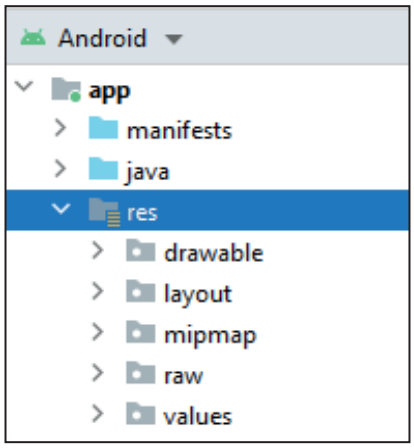
```
plugins {
    id 'com.android.application'
    id 'kotlin-android'
    id 'kotlin-android-extensions'
}
```

5- To create a directory to include the offline video files in your app, right click the **res → New → Android Resource Directory**

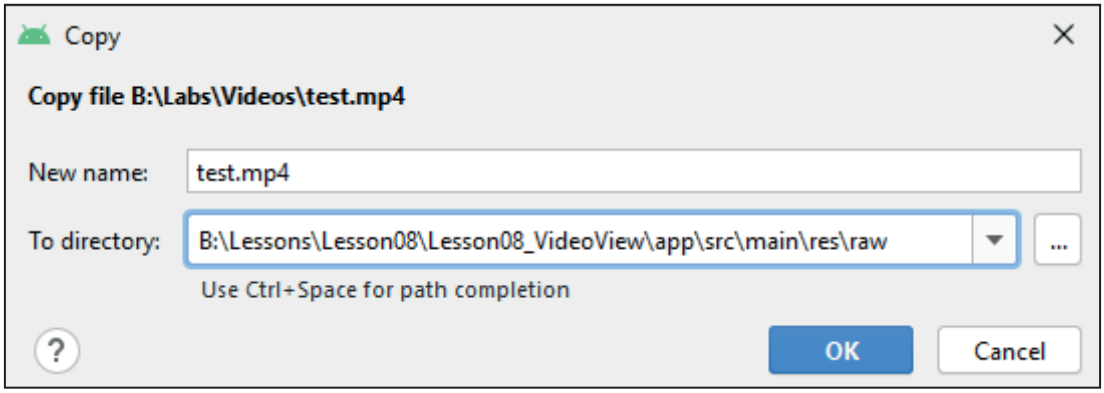
As illustrated in the figure below, select the **Resource type: raw**, and the **Directory name: raw**, then click **OK**.



You should get the following project structure:

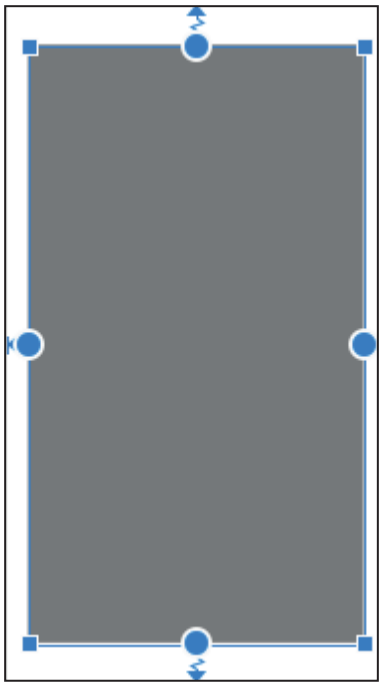


6- From the **Video** folder (**Labs → Video**), copy the **test.mp4** file (you may use any video file available on your computer), right click the **raw** directory on Android Studio, then select **Paste**. You should get the dialog box below. Click **OK**.



7- Open the **activity_main.xml** in the **Design** mode and **Delete** the "Hello World!" **TextView**.

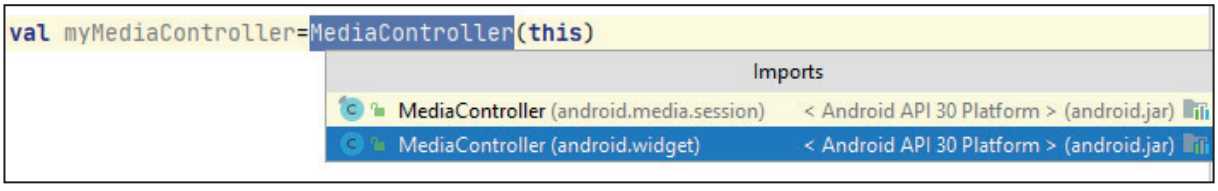
8- From the **Palette** (Palette → Widgets) panel, add the **VideoView** widget to your activity using drag and drop technique and set its constraints as illustrated in the following figure:



9- Open the **MainActivity** file and add the following code to add the media control buttons like "Play/Pause":

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val myMediaController=MediaController(this)
    }
}
```

10- Double click the **MediaController** class, click the red pop-up lamp, select **Import**, and select the **MediaController (android.widget)** choice as illustrated in the figure below:



11- Add the gray highlighted part of the following code:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val myMediaController= MediaController(this)
        myMediaController.setAnchorView(videoView)
    }
}
```

12- Configure the location of your local video file (test.mp4) which will run on your app using the following code:

```
val myLocalVideo:Uri = Uri.parse("android.
resource://$packageName/${R.raw.test}")
```

Note: If you want to run an online video file, you may use the following code:

```
val myOnlineVideo:Uri = Uri.parse("https://xxxxxxxxx")
```

13- Now add the following code to display this local video file in your app:

```
videoView.setMediaController(myMediaController)
videoView.setVideoURI(myLocalVideo)
videoView.requestFocus()
videoView.start()
```

Note: In the second line of the code above, if your video file is an online file, you should replace the **myLocalVideo** with the variable name which you have used to configure the online video URL such as **myOnlineVideo**. In this case, you must configure your app to have permission to connect to the Internet to show this online video, and to do this you must add the following XML code to your app **AndroidManifest.xml** file:

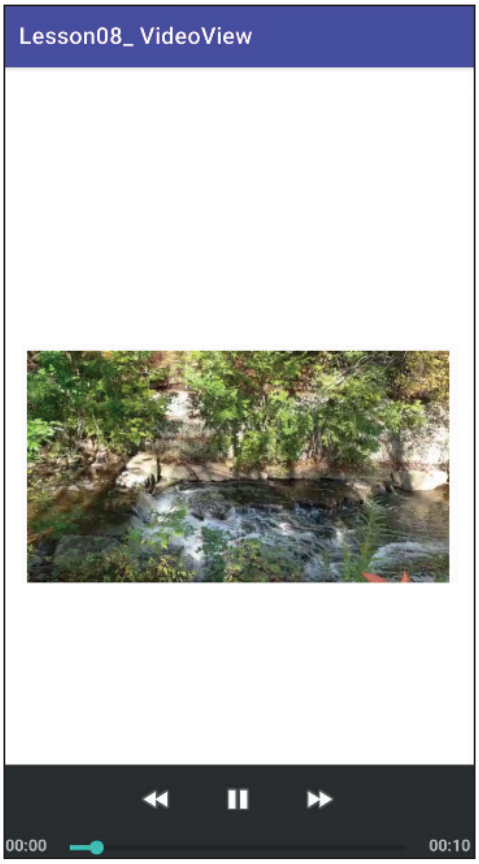
```
<uses-permission android:name="android.permission.INTERNET" />
```

The full code as follows:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
```

```
val myMediaController= MediaController(this)
        myMediaController.setAnchorView(videoView)

        val myLocalVideo:Uri = Uri.parse("android.
resource://$packageName/${R.raw.test}")

        videoView.setMediaController(myMediaController)
        videoView.setVideoURI(myLocalVideo)
        videoView.requestFocus()
        videoView.start()
    }
}
```

14- Run your app. You should get the following video view with the media controllers:



TextureView

A **`TextureView`** class or widget can be used to display a content stream such as live stream camera.

Using a TextureView is simple: all you need to do is get its **`SurfaceTexture`**. The **`SurfaceTexture`** is the image stream may come from either camera preview or video decode, and it can then be used to render content.

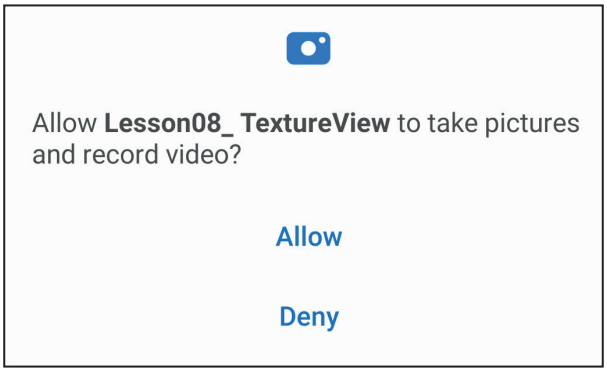
The app which including a **`TextureView`** widget can be tested only when you run your app in a hardware accelerated window (**Physical Android phone**). However, if you test a **`TextureView`** widget on a phone emulator (software), you will not get any result.

To configure your Android Studio to run your Android apps on your physical Android phone, check the last topic in lesson 4 of this course which has these configurations in details.

First, to allow your app to use your phone hardware camera, you should configure your app **AndroidManifest.xml** file (app → manifests) to ask your app user phone for permission as illustrated in the XML code below:

```
<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/
android"

    package="com.androidatc.lesson08_textureview">

    <uses-permission android:name="android.permission.CAMERA" />
```

When your app user runs this app, he/she will get the dialog box below, and the app user must tap **Allow** to open the hardware camera:



With the lab files which you have downloaded or they already exist on your computer, you should find a folder named: **Lesson08_TextureView**, this file includes Android project (Kotlin code) for this widget, and you can open this Android project from your Android Studio and run it on a connected hardware Android phone.

When you run your app, your will get a dialog box asking you for the camera permission, and after you tap Allow, your camera will run automatically. If it does not run, tap your phone Home button, then check your phone apps, you will find this app is installed on your phone. Run it from your phone.

You can write the code in different ways, and there are a lot of updates for camera operations Java code. Therefore, try to always check developer.android.com web site, or stackoverflow.com web site to find the latest code updates and solutions.

The images below for run: **Lesson08_TextureView** app on a Samsung physical phone connected with Android Studio: