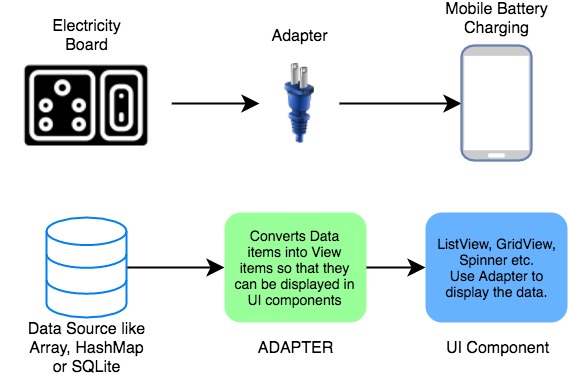
**Adapter and Adapter View**

Adapter and Adapter View are so popular, that every time we see any app with a List of items or Grid of items, we can say for sure that it is using Adapter and Adapter View.

Generally, when we create any List or Grid of data, we think we can use a loop to iterate over the data and then set the data to create the list or grid.

But what if the data is a set of 1 million products. Then using a loop will not only consume a lot of time, making the app slow, also it might end up eating all the runtime memory.

All these problems are solved by using Adapter and Adapter View.



Adapter View, is a View object, and can be used just like we use any other interface widget. The only catch here is, that it needs an Adapter to provide content to it as it is incapable of displaying data on its own.

**What is an Adapter?**

An adapter acts like a bridge between a data source and the user interface. It reads data from various data sources, coverts it into View objects and provide it to the linked Adapter view to create UI components.

The data source or dataset can be an Array object, a List object etc.

We can create our own Adapter class by extending the BaseAdapter class, which is the parent class for all other adapter class. Android SDK also provides some ready-to-use adapter classes, such as ArrayAdapter, SimpleAdapter etc.

**What is an Adapter View?**

An Adapter View can be used to display large sets of data efficiently in form of List or Grid etc, provided to it by an Adapter.

**When we say efficiently, what do we mean?**

An Adapter View is capable of displaying millions of items on the User Interface, while keeping the memory and CPU usage very low and without any noticeable lag. Different Adapters follow different strategies for this, but the default Adapter provided in Android SDK follow the following tricks:

1. It only renders those View objects which are currently on-screen or are about to some on-screen. Hence no matter how big your data set is, the Adapter View will always load only 5 or 6 or maybe 7 items at once, depending upon the display size. Hence saving memory.
2. It also reuses the already created layout to populate data items as the user scrolls, hence saving the CPU usage.

Suppose you have a dataset, like a String array with the following contents.

String days[] = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"};

Now, what does an **Adapter** do is that it takes the data from this array and creates a View from this data and then, it gives this **View** to an **AdapterView**. The AdapterView then displays the data in the way you want.

***Note:****Adapter is only responsible for taking the data from a data source and converting it into View and then passing it to the AdapterView. Thus, it is used to manage the data. AdapterView is responsible for displaying the data.*

Therefore, you can take the data from a database or an ArrayList or any other data source and then, you can display that data in any arrangement. You can display it vertically (ListView), or in rows and columns (GridView), or in drop-down menu (Spinners), etc.

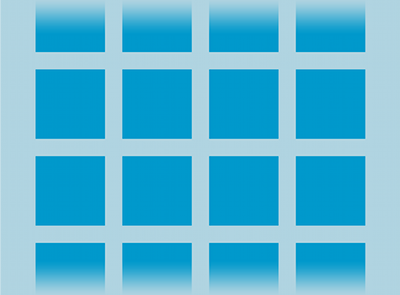
There are different types of AdapterViews. Let's have a look at some of them:

**ListView**



It displays a vertically-scrollable collection of views, where each view is positioned immediately below the previous view in the list.

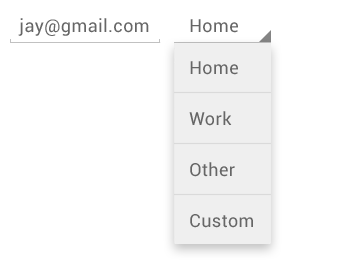
**GridView**



GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid.

**Spinner**

Spinners provide a quick way to select one value from a set of values. Touching the spinner displays a dropdown menu with all other available values, from which the user can select a new one.



Every AdapterView uses some approach for using the Adapter. We will discuss this approach in the respective AdapterView tutorials.

## ListView in Android

ListView is used when you have to show items in a vertically scrolling list. Best example of it is our device's **Contact** List. With ListView, user can easily browse the information, while scrolling up and down. You can set divider between every item and set its height and color as per your UI design.

Inside a ListView, we can show list of Text items by using TextView, or pictures using ImageView, or any other view or a combination of views.

As ListView is generally used to display a large set of data, hence it is not feasible to manually create list items for the complete data, hence Android provides us with special Adapter classes which can be used to supply data from datasets to ListView.

Following are some of the main attributes that are most commonly used:

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| android:divider | Using this attribute we can specify a divider between List items. A drawable or any color can be specified as a value for this attribute. |
| android:dividerHeight | Used to specify height of the divider. |

Below we have shown how we can add a ListView to our android app using the layout XML.

<ListView

android:id="@+id/listView"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:divider="@android:color/black"

android:dividerHeight="1dp"/>

#### Output Screen



### Using Adapter with ListView

Let's see how we can use an Adapter to read data from an array, and display it in form of a List.

We will define a ListView in the main layout XML file **activity\_main.xml**.

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

<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android" xmlns:app="http://schemas.android.com/apk/res-auto" xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:background="#FFEB3B"

tools:context="com.example.android.studytonightandroid.MainActivity">

<ListView

android:id="@+id/listView"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:divider="@android:color/black"

android:dividerHeight="1dp"/>

</android.support.constraint.ConstraintLayout>

So by doing this, we have defined a ListView to be created in our MainActivity, but what data it will show? and in what format? Where will we declare and define that?

As we specified in the [last tutorial](https://www.studytonight.com/android/adapter-and-adapter-view), that an Adapter is used to covert data items into view objects which can be used to display as UI components.

So we need a **dataset** and a **View** into which the dataset will be converted by the Adapter.

Here we have a simple Array with festivals names in it:

String[] festivals = {

"Diwali",

"Holi",

"Christmas",

"Eid",

"Baisakhi",

"Halloween"

};

As our data set has simple text values, so we can define a simple TextView to hold these values and populate the ListView. Does it sound confusing? Let it sink in.

If our dataset would have had, an image and some text along with it, then we can also define a TextView along with an ImageView to display the data in the List.

So now we will create a new XML file, with name **list\_item.xml** in the layout folder, and add a TextView in it like this,

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

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

android:id="@+id/textView"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:textStyle="bold"

android:layout\_marginLeft="10dp"

android:layout\_marginTop="5dp"

android:padding="4dp"

android:textColor="#000000"

/>

Now its time for the finale, below we have the MainActivity.java class, in which we have used an ArrayAdapter to create text views from the data in the array, and create a list by supplying those view objects to the ListView.

package listview.studytonightexample.com.listview;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

import android.view.View;

import android.widget.AdapterView;

import android.widget.ArrayAdapter;

import android.widget.ListView;

import android.widget.TextView;

import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

ListView listView;

TextView textView;

String[] festivals = {

"Diwali",

"Holi",

"Christmas",

"Eid",

"Baisakhi",

"Halloween"

};

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

listView = (ListView)findViewById(R.id.listView);

textView = (TextView)findViewById(R.id.textView);

final ArrayAdapter adapter = new ArrayAdapter(this,

android.R.layout.list\_item, android.R.id.textView, festivals);

listView.setAdapter(adapter);

listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {

@Override

public void onItemClick(AdapterView<?> adapterView, View view, int position, long l) {

// TODO Auto-generated method stub

/\* appending Happy with festival name \*/

String value = "Happy " + adapter.getItem(position);

/\* Display the Toast \*/

Toast.makeText(getApplicationContext(), value, Toast.LENGTH\_SHORT).show();

}

});

}

}

#### Output Screen



## GridView in Android

GridView just works like ListView, the only difference is that GridView is used to display grid of View objects.

The view objects can be a Text view, an Image view or a view group which has both an image and some text. Best example for this view is phone gallery which shows images in a grid. We can set number of **columns** to specific number and auto-fit the images into the columns.

Vertical and horizontal spacing between every single items of gridView can be set by verticalSpacing and horizontalSpacing.

<GridView

android:id="@+id/gridView"

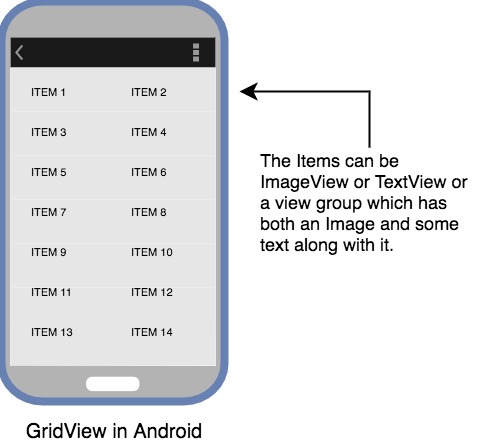
android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:verticalSpacing="2dp"

android:horizontalSpacing="2dp"

android:numColumns="2"/>



### Using Adapter with GridView

In the [last tutorial](https://www.studytonight.com/android/android-listview) we explained how an Adapter converts data set onto view objects, which are then populated in AdapterView to create the UI components. Now our adapter view is GridView. You can refer the last tutorial for detailed explanations.

We will define a ListView in the main layout XML file activity\_main.xml.

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

<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android" xmlns:app="http://schemas.android.com/apk/res-auto" xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:background="#FFEB3B"

tools:context="com.example.android.studytonightandroid.MainActivity">

<GridView

android:id="@+id/gridView"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:verticalSpacing="10dp"

android:horizontalSpacing="10dp"

android:numColumns="2"/>

</android.support.constraint.ConstraintLayout>

So for this, let's take an array of some amazing car brands.

String[] carBrands = {

"Ferrari",

"McLaren",

"Jaguar",

"Skoda",

"Suzuki",

"Hyundai",

"Toyota",

"Renault",

"Mercedes",

"BMW",

"Ford",

"Honda",

"Chevrolet",

"Volkswagon",

};

As our Grid will have only text values, hence we must define a TextView.

So now we will create a new XML file, with name **grid\_item.xml** in the layout folder, and add a TextView in it like this,

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

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

android:id="@+id/textView"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:textStyle="bold"

android:layout\_marginLeft="10dp"

android:layout\_marginTop="5dp"

android:padding="4dp"

android:textColor="#000000"

/>

And the **MainActivity.java** file will look like this:

package listview.studytonightexample.com.listview;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

import android.view.View;

import android.widget.AdapterView;

import android.widget.ArrayAdapter;

import android.widget.ListView;

import android.widget.TextView;

import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

GridView gridView;

TextView textView;

String[] carBrands = {

"Ferrari",

"McLaren",

"Jaguar",

"Skoda",

"Suzuki",

"Hyundai",

"Toyota",

"Renault",

"Mercedes",

"BMW",

"Ford",

"Honda",

"Chevrolet",

"Volkswagon",

};

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

gridView = (GridView)findViewById(R.id.gridView);

textView = (TextView)findViewById(R.id.textView);

final ArrayAdapter adapter = new ArrayAdapter(this,

android.R.layout.grid\_item, android.R.id.textView, carBrands);

listView.setAdapter(adapter);

listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {

@Override

public void onItemClick(AdapterView<?> adapterView, View view, int position, long l) {

// TODO Auto-generated method stub

/\* appending I Love with car brand names \*/

String value = "I Love " + adapter.getItem(position);

/\* Display the Toast \*/

Toast.makeText(getApplicationContext(), value, Toast.LENGTH\_SHORT).show();

}

});

}

}

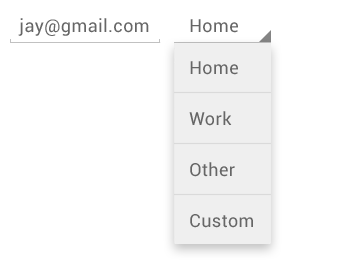
#### Output Screen



**Spinner in Android**

In this tutorial we are going to learn and explore about Spinner in Android. So let's start by understanding what is a Spinner?

A **Spinner** is a type of view that hold items in form of a dropdown menu available for user selection. It creates a menu with multiple options where a user can select any one option. Following is an example of a spinner.



We can create a Spinner, by adding the following code to our layout XML file:

<Spinner

android:id="@+id/days\_spinner"

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content" />

**What's the difference between Spinner and ListView?**

Both ListView and Spinner looks very similar. But they differ from each other.

**Spinners** provide a quick way to select one value from a given set of values and in the default state, a spinner only shows the currently selected value. When you touch(tap on) the spinner, it displays a dropdown menu with all other available values(options), from which the user can select a new one.

**ListView**, on the other hand, is a view group that displays a list of scrollable items. The list items are automatically inserted to the list using an Adapter that pulls content from a data source such as an array or database table and converts each dataitem into a view that is placed in the list.

Therefore, a Spinner and ListView differs in the way they appear and in their usage too. If you want to select only one value from a set of options then you should use a Spinner. If you want to display a list of data then use a list view.

**How does a Spinner works?**

1. **Define the data source**

There should be a data source that will be used to display the data in the spinner.

String days[] = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"};

A data source can be a list, an array, a JSON data or data coming from a database.

1. **Define the Adapter**

There will be an Adapter(ArrayAdapter etc) to take data from data source, create a view of it and then pass it to the AdapterView i.e. **Spinner**. Thus, you need to tell the Adapter how to display the data by specifying a layout for the drop down.

ArrayAdapter<String> adapter = new ArrayAdapter<>(this, android.R.layout.simple\_spinner\_item, days);

There are 3 parameters used to instantiate an ArrayAdapter(we have used an ArrayAdapter in our example):

* + **Context c**: Refers to the current class object where ArrayAdapter is being instantiated.
  + **Layout**: It is a layout file that defines how a single item will appear in the spinner. Android SDK by default provides layouts like simple\_spinner\_item and simple\_spinner\_dropdown\_itemand if you do not have any special design requirements, we suggest you use these only.
  + **Data source**: It is the data source from which data will be converted into a view.

1. **Define any actions for Spinner**

To add an event handler for option selection, you can implement [OnItemSelectedListener](https://developer.android.com/reference/android/widget/AdapterView.OnItemSelectedListener.html" \t "_blank) interface, to determine what happens when a user selects any option from the menu. For this, you need to implement the above mentioned interface and override two methods:

**onItemSelected()**

This method has 4 parameters:

* + **AdapterView av**: It's the Spinner view that you have used.
  + **View v**: It defines the TextView inside the spinner that was clicked.
  + **int position**: It tells the position of the item that was clicked in the Spinner. The index or the position starts from 0.
  + **long id**: It gives the row id of the item clicked in the Spinner. This parameter is mainly used when dealing with databases in Android.

**onNothingSelected()**

This method has only 1 parameter:

**AdapterView av**: It's the Spinner view that you have used. This method is called whenever the currently selected item is removed from the list of available items in the Spinner. If the adapter is modified such that the currently selected item is no longer available, then this method will be called. This method may be used so that you can set which item will be selected when the previous item is no longer available. This prevents the spinner from automatically selecting the next item in the list.

If mySpinner is an instance of a Spinner view then, following is how we can implement the above listeners:

mySpinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {

@Override

public void onItemSelected(AdapterView<?> parent, View view, int position, long id)

{

// An item was selected. You can retrieve the selected item using

// parent.getItemAtPosition(pos)

}

@Override

public void onNothingSelected(AdapterView<?> parent)

{

// If an option is removed then what to do

// or anything else

}

});

We hope you have got an idea of what Spinners are and how they can be implemented. In the next tutorial, we will see how we can create a Spinner in our Android Application.

**Working with Spinners in Android**

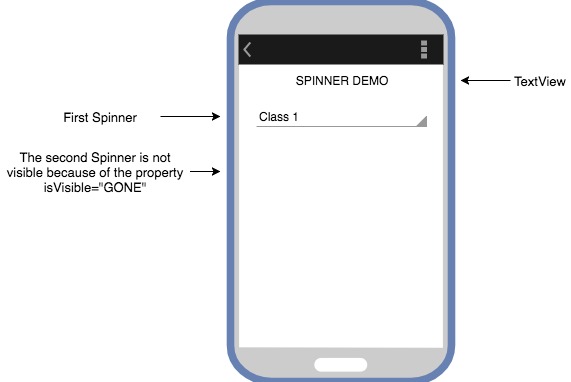
You have already studied about what Spinners are and how they work. So let's checkout an example and learn how we can implement it in our Android App.

In this tutorial, we are going to create an application that works with two Spinners, which are inder-dependent. The main layout of the application will contain our **one TextView** and **two spinners** as its child component views. So we need to customize the layout(shown in below image) where one Spinner is placed below the TextView and the other spinner is below the first spinner.

As the options inside the second Spinner will depend on what we select in the first Spinner, hence initially the second spinner will be hidden and it will only appear when user has selected one option from the first spinner.

The second spinner has the property isVisible equals to **GONE**, which means it exists in the layout but it will not be visible(or it will be hidden).

Hence, in the image below, it looks like it's not there. But you can try and change the value of isVisible to VISIBLE to see if it is there or not. But then change it back to GONE for now.



The first Spinner(classSpinner) holds the list of classes(in school) to be selected by the user and based on that choice we will assign the values to the second Spinner(divSpinner). When user selects an optin from the second spinner too, then we will create a Toast and display the chosen values on screen.

So let's start, here we have the layout XML file. If you want, copy paste the below XML to start with this example in your local machine.

**main\_activity.xml**

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

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

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context="com.example.akshay.studytonightandroid.MainActivity">

<TextView

android:id="@+id/tvDemo"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:layout\_alignParentStart="true"

android:layout\_alignParentTop="true"

android:gravity="center"

android:text="SPINNER DEMO"

android:layout\_alignParentLeft="true" />

<Spinner

android:id="@+id/classSpinner"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:layout\_below="@+id/tvDemo"

android:layout\_marginTop="25dp"

android:entries="@array/items\_class"/>

<Spinner

android:id="@+id/divSpinner"

android:visibility="gone"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:layout\_below="@id/classSpinner"

android:layout\_toLeftOf="@id/classSpinner"

android:layout\_marginTop="10dp"

/>

</RelativeLayout>

**NOTE:** If you are getting an error in this, that is because you have not yet created the below data XML file. Once you will create that, the error in this file will be resolved.

As you can see, we have one TextView and two Spinner views inside the xml file along with a few properties specified.

To add items to the Spinner, there are two possible ways to provide it with a set of options. One by declaring an array and defining the items in it. But for this example, we will try the other way i.e. using an XML.

We can define a string array in XML like we have shown below in the **strings.xml** file. In Android, we must put the data XML files like our **strings.xml** file in **app → res → values → strings.xml**.

**strings.xml**

<resources>

<string name="app\_name">StudytonightAndroid</string>

<string-array name="items\_class">

<item>Class 1</item>

<item>Class 2</item>

<item>Class 3</item>

<item>Class 4</item>

</string-array>

<string-array name="items\_div\_class\_1">

<item>Div 1-A</item>

<item>Div 1-B</item>

<item>Div 1-C</item>

<item>Div 1-D</item>

</string-array>

<string-array name="items\_div\_class\_2">

<item>Div 2-A</item>

<item>Div 2-B</item>

<item>Div 2-C</item>

<item>Div 2-D</item>

</string-array>

<string-array name="items\_div\_class\_3">

<item>Div 3-A</item>

<item>Div 3-B</item>

<item>Div 3-C</item>

<item>Div 3-D</item>

</string-array>

<string-array name="items\_div\_class\_4">

<item>Div 4-A</item>

<item>Div 4-B</item>

<item>Div 4-C</item>

<item>Div 4-D</item>

</string-array>

</resources>

By now, we are done with the following things:

1. We have understood the design that we are creating in our Android App, which will have 1 TextView and 2 Spinners.
2. We have defined the layout XML for the User interface.
3. We have also defined our dataset for the Spinners.

In our dataset, string-array named **items\_class** will be assigned to the classSpinner to display the class items in the dropdown list.

To add these entries to the spinner all we have to do is add a property android:entries="@array/items\_class" in the main activity layout XML file. Doing so will assign the values present in the items\_class array to the classSpinner.

As per the user's selection of the option from the first spinner, our second Spinner will appear along with options based on the first spinner's selection. We will do this through Java code as the user will select from the first dropdown at run time i.e. dynamically.

To handle the GUI events, we need to implement the code inside the **MainActivity.java** file. We will start by creating instances of both the spinners and then, assign the setOnItemSelectedListener()on both the spinners.

**MainActivity.java**

public class MainActivity extends AppCompatActivity

{

// these are the global variables

Spinner classSpinner, divSpinner;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

classSpinner = (Spinner) findViewById(R.id.classSpinner);

divSpinner = (Spinner) findViewById(R.id.divSpinner);

// Class Spinner implementing onItemSelectedListener

classSpinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener()

{

@Override

public void onItemSelected(AdapterView<?> parent, View view, int position, long id)

{

// do something upon option selection

}

@Override

public void onNothingSelected(AdapterView<?> parent)

{

// can leave this empty

}

});

// Div Spinner implementing onItemSelectedListener

divSpinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener()

{

@Override

public void onItemSelected(AdapterView<?> parent, View view, int position, long id)

{

// do something upon option selection

}

@Override

public void onNothingSelected(AdapterView<?> parent)

{

// can leave this empty

}

});

}

}

**Changing Value of second Spinner based on first Spinner value**

Inside the onItemSelected() method of classSpinner, you have to get the selected item from the dropdown list and based on that value, you have to assign entries i.e. options to divSpinner from the string-array resource. So add the following code inside classSpinner's onItemSelected() method.

classSpinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener()

{

@Override

public void onItemSelected(AdapterView<?> parent, View view, int position, long id)

{

// Get Selected Class name from the list

String selectedClass = parent.getItemAtPosition(position).toString();

switch (selectedClass)

{

case "Class 1":

// assigning div item list defined in XML to the div Spinner

divSpinner.setAdapter(new ArrayAdapter<String>(MainActivity.this,

android.R.layout.simple\_spinner\_dropdown\_item,

getResources().getStringArray(R.array.items\_div\_class\_1)));

break;

case "Class 2":

divSpinner.setAdapter(new ArrayAdapter<String>(MainActivity.this,

android.R.layout.simple\_spinner\_dropdown\_item,

getResources().getStringArray(R.array.items\_div\_class\_2)));

break;

case "Class 3":

divSpinner.setAdapter(new ArrayAdapter<String>(MainActivity.this,

android.R.layout.simple\_spinner\_dropdown\_item,

getResources().getStringArray(R.array.items\_div\_class\_3)));

break;

case "Class 4":

divSpinner.setAdapter(new ArrayAdapter<String>(MainActivity.this,

android.R.layout.simple\_spinner\_dropdown\_item,

getResources().getStringArray(R.array.items\_div\_class\_4)));

break;

}

//set divSpinner Visibility to Visible

divSpinner.setVisibility(View.VISIBLE);

}

@Override

public void onNothingSelected(AdapterView<?> parent)

{

// can leave this empty

}

});

We can get the value of the selected class name from the ClassSpinner. For that we have used parent.getItemAtPosition(position).toString() method. Here, parent is the classSpinner view, and position is the position of the option selected.

Hence in parent.getItemAtPosition(position).toString() we are fetching the item(option) of the classSpinner dropdown present at the position which is stored in the parameter position.

Once we have the name of the class selected in the first dropdown, we can use a simple switch case to initialise the second dropdown with appropriate set of values.

We will initialise an ArrayAdapter with the respective string-array and will ste the adapter to our second spinner divSpinner.

To access the string-array from string resource in XML, we will have to use the method, getResources().getStringArray(ID\_OF\_ARRAY)

And at the end, we will set the visibility of the second spinner to be **VISIBLE**.

**Creating a Toast to show the selcted values**

Next, when the user selects an option from the second Spinner, we will create a Toast and display the selected values on screen.

// getting selected value from second spinner

selectedDiv = parent.getItemAtPosition(position).toString();

// creating Toast

Toast.makeText(MainActivity.this, "\n Class: \t " + selectedClass + "\n Div: \t" + selectedDiv, Toast.LENGTH\_LONG).show();

**Complete Code for MainActivity.java**

Below is the complete code for MainActivity.java

package com.example.android.studytonight;

import android.os.Bundle;

import android.support.v7.app.AppCompatActivity;

import android.view.View;

import android.widget.AdapterView;

import android.widget.ArrayAdapter;

import android.widget.Spinner;

import android.widget.Toast;

public class MainActivity extends AppCompatActivity

{

// these are the global variables

Spinner classSpinner, divSpinner;

// string variable to store selected values

String selectedClass, selectedDiv;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

classSpinner = (Spinner) findViewById(R.id.classSpinner);

divSpinner = (Spinner) findViewById(R.id.divSpinner);

// Class Spinner implementing onItemSelectedListener

classSpinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener()

{

@Override

public void onItemSelected(AdapterView<?> parent, View view, int position, long id)

{

String selectedClass = parent.getItemAtPosition(position).toString();

switch (selectedClass)

{

case "Class 1":

// assigning div item list defined in XML to the div Spinner

divSpinner.setAdapter(new ArrayAdapter<String>(MainActivity.this,

android.R.layout.simple\_spinner\_dropdown\_item,

getResources().getStringArray(R.array.items\_div\_class\_1)));

break;

case "Class 2":

divSpinner.setAdapter(new ArrayAdapter<String>(MainActivity.this,

android.R.layout.simple\_spinner\_dropdown\_item,

getResources().getStringArray(R.array.items\_div\_class\_2)));

break;

case "Class 3":

divSpinner.setAdapter(new ArrayAdapter<String>(MainActivity.this,

android.R.layout.simple\_spinner\_dropdown\_item,

getResources().getStringArray(R.array.items\_div\_class\_3)));

break;

case "Class 4":

divSpinner.setAdapter(new ArrayAdapter<String>(MainActivity.this,

android.R.layout.simple\_spinner\_dropdown\_item,

getResources().getStringArray(R.array.items\_div\_class\_4)));

break;

}

//set divSpinner Visibility to Visible

divSpinner.setVisibility(View.VISIBLE);

}

@Override

public void onNothingSelected(AdapterView<?> parent)

{

// can leave this empty

}

});

// Div Spinner implementing onItemSelectedListener

divSpinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener()

{

@Override

public void onItemSelected(AdapterView<?> parent, View view, int position, long id)

{

selectedDiv = parent.getItemAtPosition(position).toString();

/\*

Now that we have both values, lets create a Toast to

show the values on screen

\*/

Toast.makeText(MainActivity.this, "\n Class: \t " + selectedClass +

"\n Div: \t" + selectedDiv, Toast.LENGTH\_LONG).show();

}

@Override

public void onNothingSelected(AdapterView<?> parent)

{

// can leave this empty

}

});

}

}

When we run our android application, we will get the first toast message by default because first item is already assigned to the spinner. Don't worry about it. Just go and try to change the spinner selection for class and division dropdowns and you should see the results as below images.

When we select Class option, as per the class selection our app will load other items in the DivSpinner and we can then select items from the second spinner. When we select an item from the second spinner, our app will display selected items on device screen.

