CS 473 – MDP
Mobile Device Programming

© 2020 Maharishi International University

Maharishi International University

CS 473 – MDP
Mobile Device Programming

MS.CS Program

Department of Computer Science

Renuka Mohanraj, Ph.D.,

CS 473 – MDP
Mobile Device Programming

Lesson-6
User Input Controls – Day 1

Maharishi International University

Wholeness

- Your app's user interface is everything that the user can see and interact with. Android provides a variety of pre-built UI components such as TextView, Button, AutoCompleteTextView and various list views that allows you to build the graphical user interface for your app. Android also provides other UI modules for special interfaces such as dialogs, notifications, and menus. *The ultimate provider of tools for the creation of beautiful and functional content is pure intelligence itself; all creativity arises from this field's self-interacting dynamics.*

Agenda

- Day 1
  - Input Controls- Text and Buttons
  - AutoCompleteTextView
  - Toast Message
  - Spinner
  - ListView
  - Customized GridView
  - Dialogs – Alert, DatePicker, Time Picker
  - Hands on Examples
- Day 2
  - Recycler View and CardView

Users expect to interact with apps

- Clicking, pressing, talking, typing, and listening

- Using user input controls such buttons, menus, keyboards, text boxes, and a microphone

- Navigating between activities

User interaction design

Important to be obvious, easy, and consistent:

- Think about how users will use your app

- Minimize steps

- Use UI elements that are easy to access and understand

- Follow Android best practices

  - https://developer.android.com/distribute/best-practices
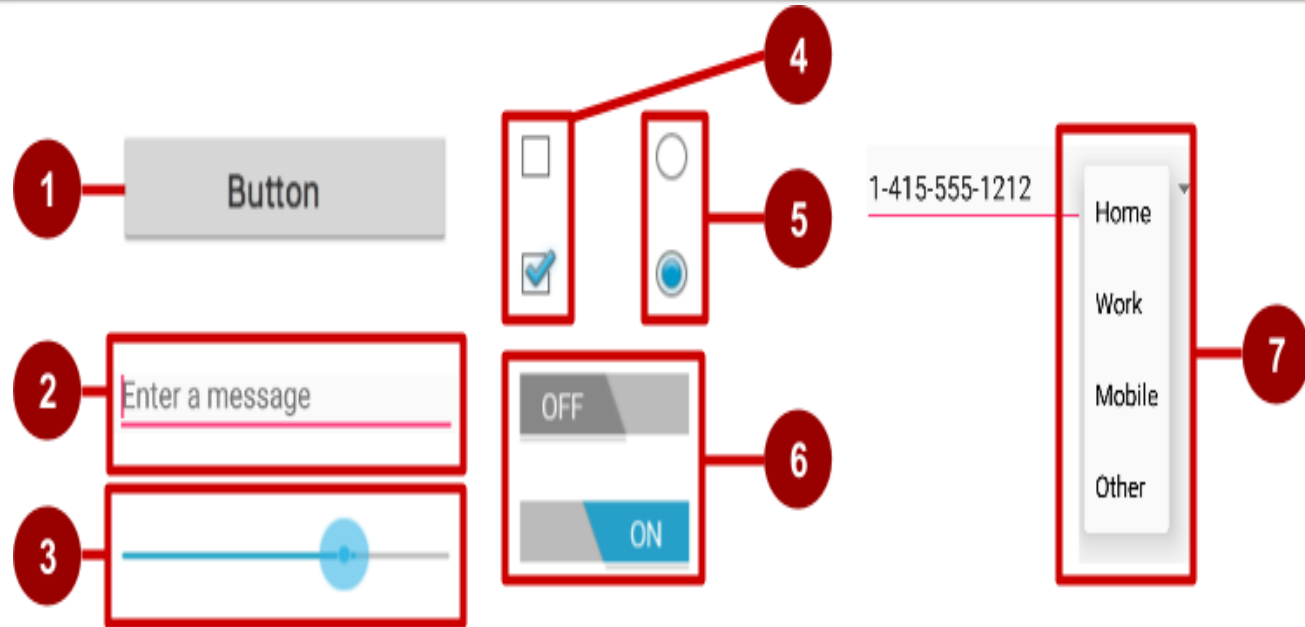
- Meet user's expectations

Ways to get input from the user

- Free form
  - Text and voice input

- Actions
  - Buttons
  - Contextual menus
  - Gestures
  - Dialogs

- Constrained choices
  - Pickers
  - Checkboxes
  - Radio buttons
 - Toggle buttons
  - Spinners

# Examples of user input controls

1. Button
2. Text field
3. Seek bar
4. Checkboxes
5. Radio buttons
6. Toggle
7. Spinner

- The **View** class is the basic building block for all UI components, including input controls

- View is the base class for classes that provide interactive UI components

- View provides basic interaction through `android:onClick`

Alert dialog, date picker, time picker
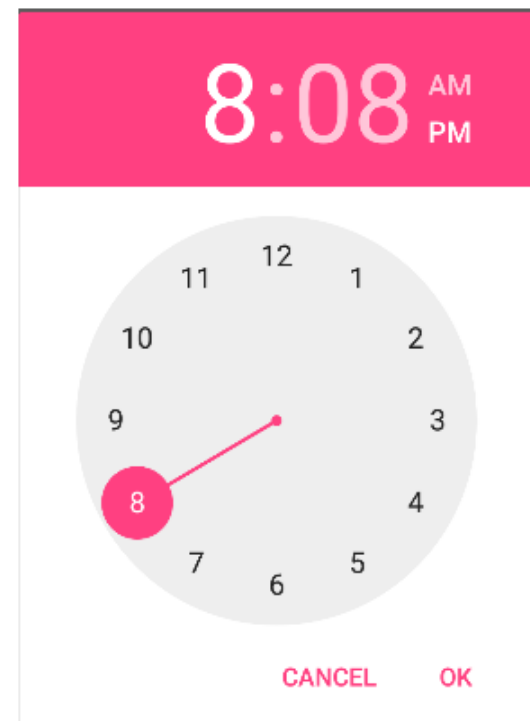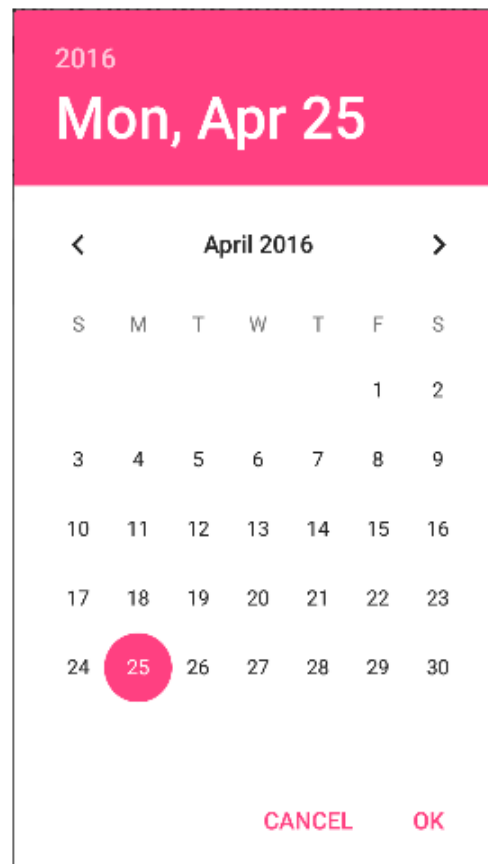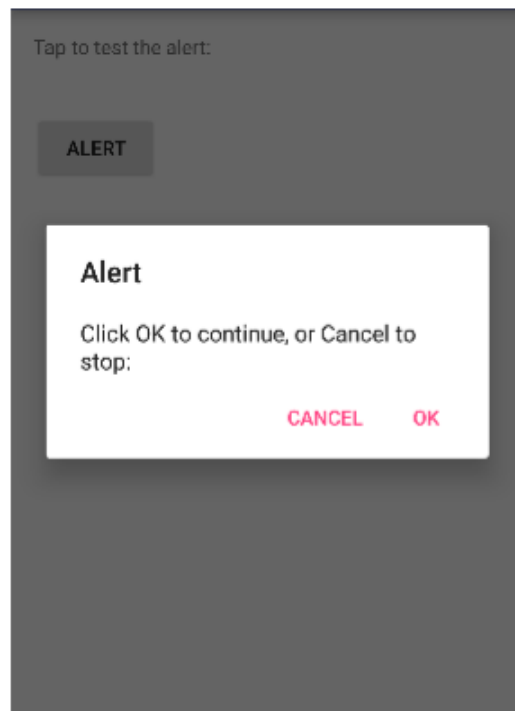
EditText

- <u>EditText</u> class used to accept inputs from the user
- Multiple lines of input
- Characters, numbers, and symbols
- Spelling correction
- Tapping the Return (Enter) key starts a new line
- Customizable
- Get the EditText object for the EditText view
  - ```
    EditText simpleEditText =
        (EditText) findViewById(R.id.edit_simple);
    ```
- Retrieve the CharSequence and convert it to a string
  - ```
    String strValue =
        simpleEditText.getText().toString();
    ```

Common input types

- textShortMessage—Limit input to 1 line
- textCapSentences—Set keyboard to caps at beginning of sentences
- textAutoCorrect—Enable autocorrecting
- textPassword—Conceal typed characters
- textEmailAddress—Show an @ sign on the keyboard
- phone—numeric keyboard for phone numbers
- textPersonName – allow single line of Input

Example

```
android:inputType="phone"

android:inputType="textAutoCorrect|textCapSentences"
```

Making Choices

- Checkboxes

- [x] Chocolate Syrup
- [x] Sprinkles
- [ ] Crushed Nuts

- Radio buttons

Choose a delivery method:
- (●) Same day messenger service
- ( ) Next day ground delivery
- ( ) Pick up

- Spinner

1-415-555-1212
- Home
- Work
- Mobile
- Other

- Toggles

Turn on or off: **ON**   Turn on or off: **OFF**

- Switch

Turn on or off: Turn on or off:
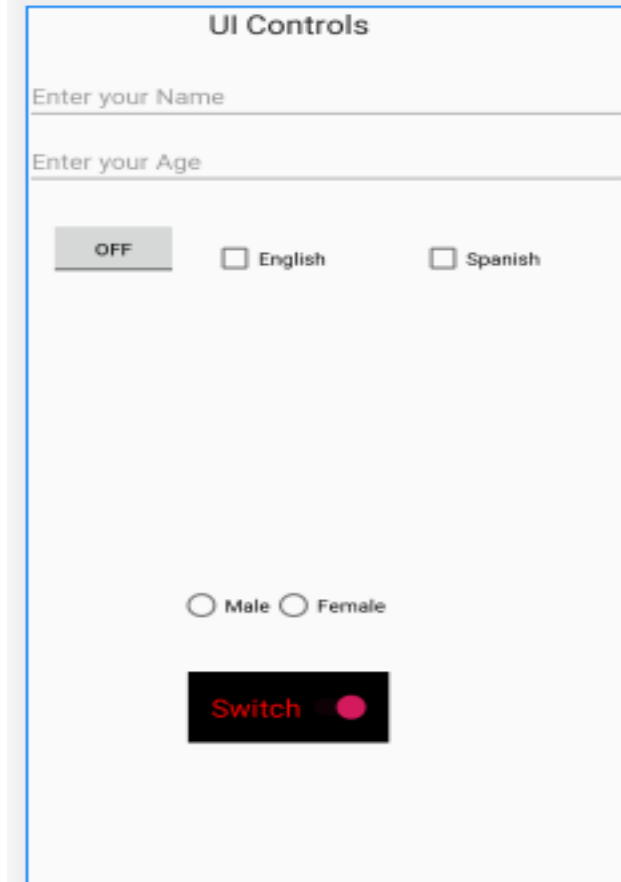
Radio buttons

- User can select one of several choices
- Put radio buttons in a RadioGroup
- Checking one unchecks another
- Put radio buttons in a vertical list
  or horizontally if labels are short
- Every radio button can have an onClick handler
- Commonly used with a submit button
  for the RadioGroup

- Refer : UIDemo to know about UI Components and their click events.

Main Point 1

- There are number of input controls that can be used including Button, EditText, Checkbox, Radio Button, Toggle Button, Spinner and Picker to develop an app with interactive UI's. *Similarly Creative Intelligence synthesizes parts for completeness of living through the interaction of inner self by doing regular transcendental mediation.*

AutoCompleteTextView
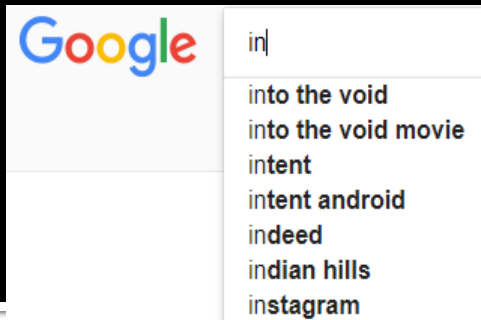
- The AutoCompleteTextView is sort of a hybrid between the EditText (field) and the Spinner. With auto-completion, as the user types, the text is treated as a prefix filter, comparing the entered text as a prefix against a list of candidates.
- Matches are shown in a selection list that folds down from the field. The user can either type out an entry (e.g., something not in the list) or choose an entry from the list to be the value of the field.
- In addition, AutoCompleteTextView has a Threshold property, to indicate the minimum number of characters a user must enter before the list filtering begins.
- To create a UI Component use <AutoCompleteTextView> tag in XML.
- To provide an auto completion support to the user first we must configure the values. Configure the values either using
  - XML approach or Kotlin code approach.

XML Approach

Step 1: In your Android Project go to the folder app→res→values→strings.xml

Set the values by using the given lines of codes

```
<string-array name="array_name">
    <item>value1</item>
    <item>value2</item>
    ………….
</string-array>
```

Step 2: Create an AutoCompleteTextView UI component in XML with an id.

Step 3: Use the following code to get the Xml configured String values into your Activity.

```
String[] values=getResources().getStringArray(R.array.array_name);
```

XML Approach

Step 4: for presenting the values we need to create an Adapter. In Android there are various types of Adapters available such as ArrayAdapter, Custom Adapter(Customizing ListView, GridView), Cursor Adapter(Work with Database) etc., In this lesson, we are using ArrayAdapter.
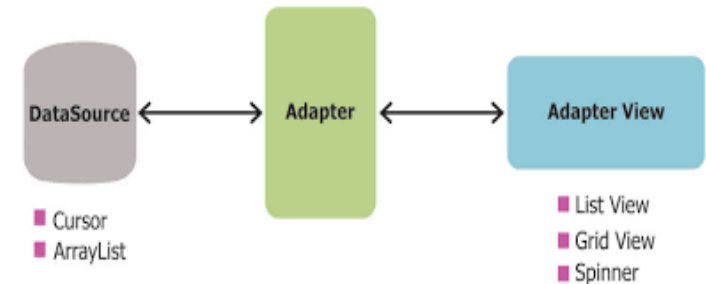
Adapter : An Adapter object acts as a bridge between an AdapterView and the underlying data for that view. The Adapter provides access to the data items. The Adapter is also responsible for making a View for each item in the data set.

var adapter= ArrayAdapter<String>(context, Layout_Resorce File, values);

actv.setAdapter(adapter); // To specify the Adapter Object. Here actv is an AutocompleteTextView object

actv.setThreshold(int); **// To specify the dropdown support, after how many characters of user entry**

Context : It allows access to application-specific resources and classes, as well as up-calls for application-level operations such as launching activities, broadcasting and receiving intents, etc. Mostly use **this or getApplicationContext().**

Enter Text :

In

India
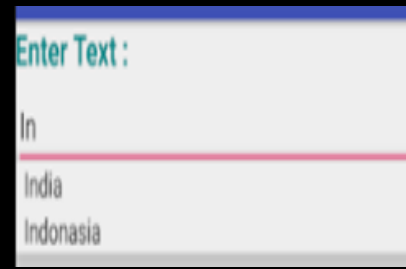
Indonasia

strings.xml

```xml
<resources>
    <string
name="app_name">ACTVDemo</string>
    <string-array name="countries">
        <item>India</item>
        <item>Indonasia</item>
        <item>USA</item>
        <item>Asia</item>
        <item>Africa</item>
        <item>Siria</item>
        <item>Sri Lanka</item>
        <item>Canada</item>
        <item>Koria</item>
        <item>Island</item>
        <item>Bangaladesh</item>
        <item>Nepal</item>
    </string-array>
</resources>
```

XML Code

```xml
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Enter Text : "
        android:textSize="20sp"
        android:textStyle="bold"
        android:textColor="#008888" />
    <AutoCompleteTextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/actv"/>
</LinearLayout>
```

AdapterView.OnItemClickListener

interface AdapterView.OnItemClickListener {
// Callback method to be invoked when an item in this AdapterView has been clicked.
onItemClick(parent: AdapterView<*>, view: View, position: Int, id: Long);
} (*) indicates like (?) wildcard in Java

Parameters	
parent	AdapterView: The AdapterView where the click happened.
view	View: The view within the AdapterView that was clicked (this will be a view provided by the adapter)
position	int: The position of the view in the adapter.
id	long: The row id of the item that was clicked.

Implementers can call getItemAtPosition(position) if they need to access the data associated with the selected item.
A position starts from 0 for ListView, If you are using an ArrayAdapter, position and id become the same, whereas to get a proper row id it is important that the cursor, which was passed to the adapter, contains unique id for each row in the table.

MainActivity.kt

```kotlin
class MainActivity : AppCompatActivity() {
    private lateinit var  strings : Array<String>
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        strings = arrayOf  ("Asia","Australia","America","Belgium","Brazil","Canada",
                            "California","Dubai","France","Paris")
    // Get the XML configured vales into the Activity and stored into an String Array
    //strings = getResources().getStringArray(R.array.countries);
    /* Pass three parameters to the ArrayAdapter
    1. The current context,
    2. The resource ID for a built-in layout file containing a TextView to use when instantiating views,
       which are available in android.R.layout
    3. The objects to represent in the values
    */
        val adapter = ArrayAdapter<String>(this, android.R.layout.simple_spinner_item, strings)
        actv.setAdapter(adapter)
        actv.threshold = 1
```

MainActivity.kt

```kotlin
    // Set an item click listener for auto complete text view
actv.onItemClickListener =
        AdapterView.OnItemClickListener {
         parent, view, position, id ->
           Toast.makeText(this,
               "Item selected is " +
               parent.getItemAtPosition(position),
               Toast.LENGTH_LONG).show()
        }
    }
}
```
Refer : ACTVDemo

Toast

- Toast is one of the notification method in Android which is used to display text on the screen for few seconds at the bottom.

Toast makeText (Context context, CharSequence msg, int duration).show()

Example

Toast.makeText(applicationContext,"Hello Toast",Toast.LENGTH_SHORT).show()
Toast.makeText(this,"Hello Toast",Toast.LENGTH_LONG).show()

Parameters	
context	Context: The context to use. Usually your <u>Application</u> or <u>Activity</u> object.
msg	CharSequence: The text to show. Can be formatted text.
duration	int: How long to display the message. Either <u>LENGTH_SHORT</u>(1sec) or <u>LENGTH_LONG</u>(3 sec)

Toast

- To print the Toast in different position use the following code.
- Use different position such as BOTTOM, LEFT, RIGHT, TOP and CENTER with x,y coordinates

```
var tst = Toast.makeText(applicationContext,
            "Top Toast",
            Toast.LENGTH_LONG)
   tst.setGravity(Gravity.TOP, 0, 0)
   tst.show()
```

Difference Between XML And Kotlin Approach for providing data resources

XML Approach
- To provide static data

  - Example : Country names are static
- To provide Multi language support-I18N-(Internationalization) -(Discussed later in Localization)

Kotlin code Approach
- To provide dynamic data

  - Example : Movies list

Spinners

- Spinners provide a quick way to select one value from a set. In the default state, a spinner shows its currently selected value.
- Touching the spinner displays a dropdown menu with all other available values, from which the user can select a new one.
- Spinners scroll automatically if necessary
- Implementing a spinner needs to follow the below steps
    1. Create Spinner UI element in the XML layout
    2. Define spinner choices in an array(Kotlin/Strings.xml resource)
    3. Create an adapter with default spinner layouts
    4. Attach the adapter to the spinner
    5. Implement `onItemSelectedListener` method

Creating Spinner

```
<Spinner
    android:id="@+id/planets_spinner"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:entries="@array/planets_array"/>
```

- Android:entries helps to retrieve the values from string resources to the xml.
- Configure the values either using XML approach or Kotlin code approach

Populate the spinner choices using Resources

Step 1: In your Android Project go to the folder
app→res→values→strings.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="planets_array">
        <item>Mercury</item>
        <item>Venus</item>
        <item>Earth</item>
        <item>Mars</item>
        <item>Jupiter</item>
        <item>Saturn</item>
        <item>Uranus</item>
        <item>Neptune</item>
    </string-array>
</resources>
```

Populate the spinner with user choices

- The choices you provide for the spinner can come from any source, but must be provided through SpinnerAdapter, such as an ArrayAdapter if the choices are available in an array or a CursorAdapter if the choices are available from a database query.
- For instance, if the available choices for your spinner are pre-determined, you can provide them with a string array defined in a string resource file:

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string-array name="planets_array">
      <item>Mercury</item>
      <item>Venus</item>
      <item>Earth</item>
      <item>Mars</item>
      <item>Jupiter</item>
      <item>Saturn</item>
      <item>Uranus</item>
      <item>Neptune</item>
   </string-array>
</resources>
```

Responding to user selections

- When the user selects an item from the drop-down, the Spinner object receives an on-item-selected event.
- To define the selection event handler for a spinner, implement the AdapterView.OnItemSelectedListener interface and the corresponding onItemSelected() callback method.
- The AdapterView.OnItemSelectedListener requires the onItemSelected() and onNothingSelected() callback methods.
- Then you need to specify the interface implementation by calling setOnItemSelectedListener():

Example from Demo : SpinnerDemo folder

# Hands on Example-3 for Spinner - Planet Weight Calculator
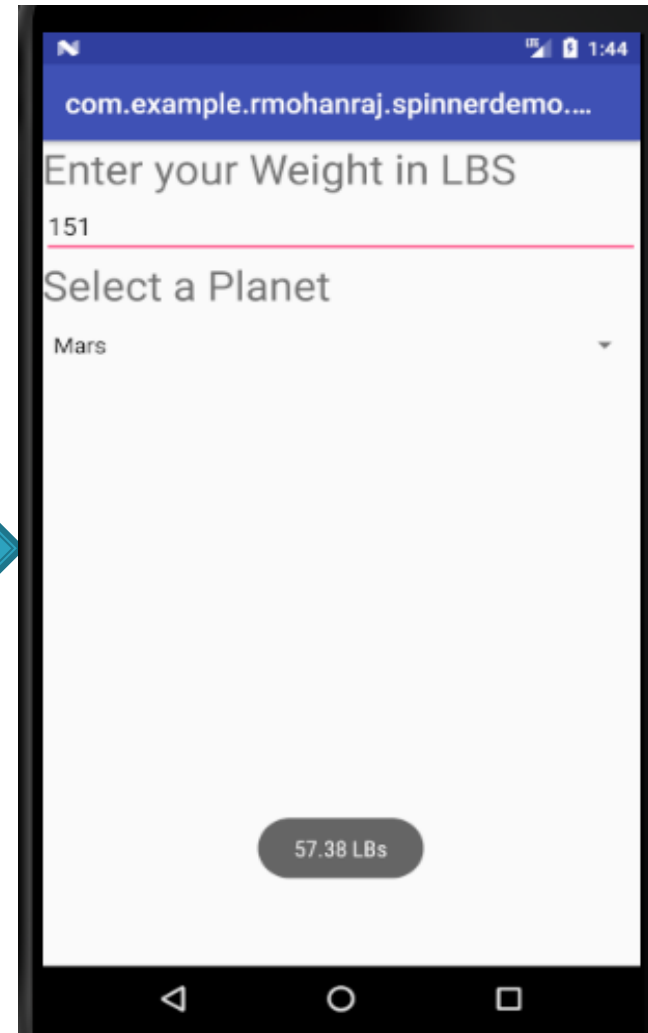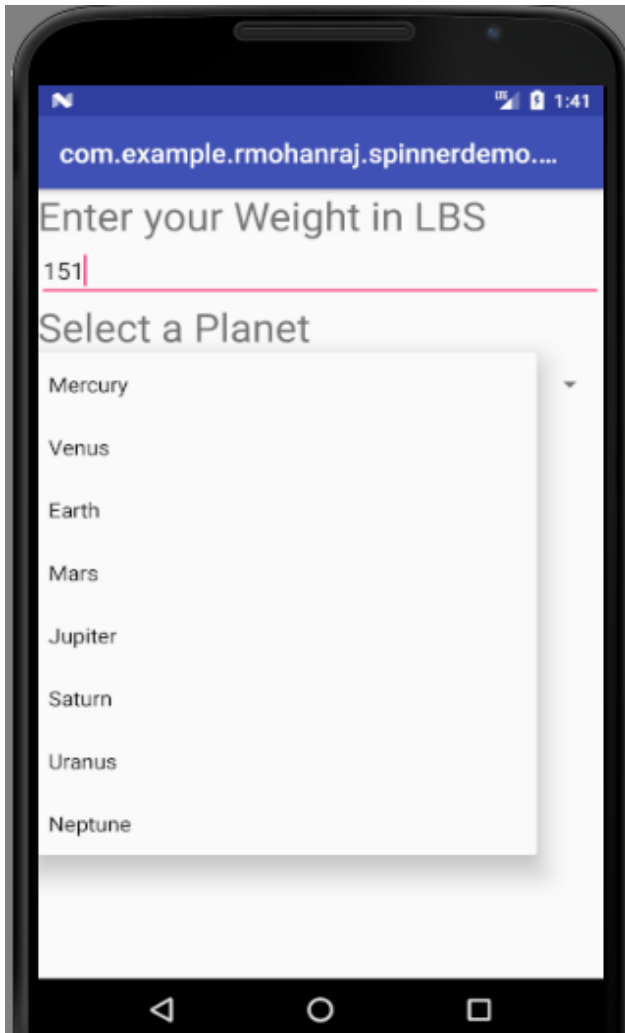
Simple ListView

- ListView is a view group that displays a list of scrollable items. An Adapter that pulls content from a source, such as, a query or an array, helps to insert the list items automatically.

- Each item result is converted into a View and added to the list by the Adapter.
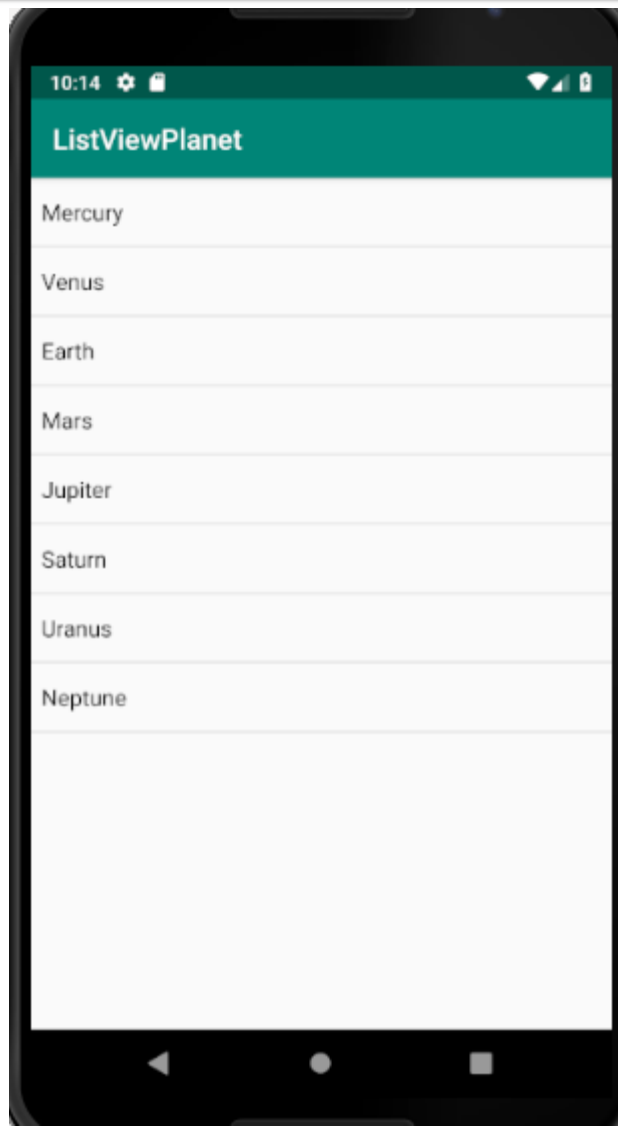
 <ListView   android:id="@android:id/list"
android:layout_width="match_parent"
android:layout_height="wrap_content"
  </ListView>

- Example : ListViewPlanet
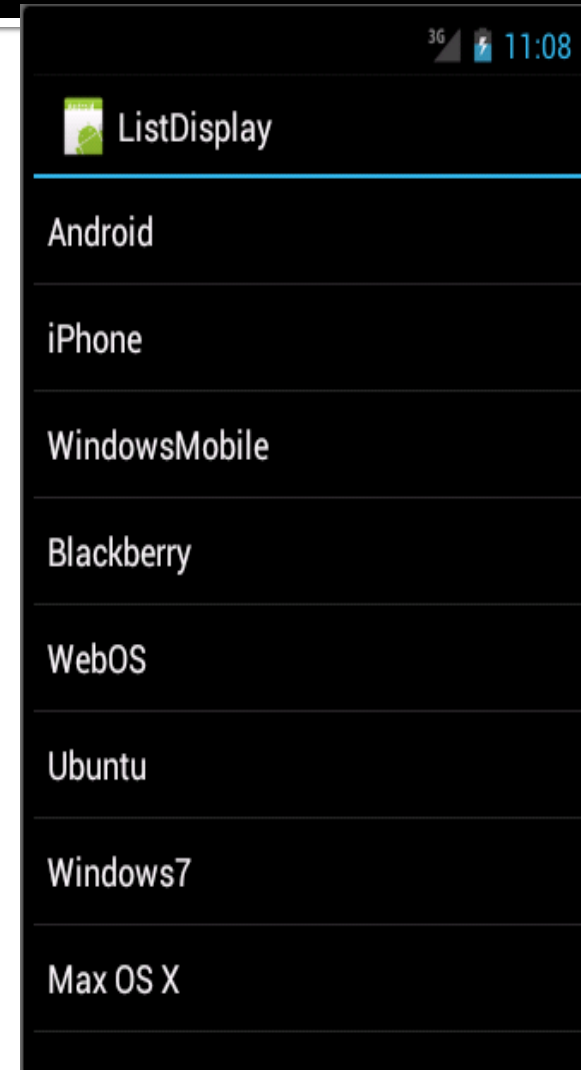
Hands on Example-4 - ListView
Planet Information

ListView Code

```kotlin
package com.example.listviewplanet
import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.ArrayAdapter
import kotlinx.android.synthetic.main.activity_main.*

class MainActivity : AppCompatActivity() {
    var imageges : IntArray  = intArrayOf(R.drawable.mercury, R.drawable.ven, R.drawable.earth, R.drawable.mars,
                                R.drawable.jupitar, R.drawable.saturn, R.drawable.uranus, R.drawable.neptune)
    var planets : Array<String>  = arrayOf<String>("Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val adapter = ArrayAdapter<String>( context: this, android.R.layout.simple_spinner_dropdown_item, planets)
        lview.adapter = adapter
        lview.setOnItemClickListener { parent, view, position, id ->

            val intent = Intent(applicationContext, PlanetActivity::class.java)
            intent.putExtra( name: "image", imageges[position])
            startActivity(intent)

        }
    }
}
```

Main Point 2

- Auto Complete View is an editable text view that shows completion suggestions automatically while the user is typing. The list of suggestions is displayed in a dropdown menu from which the user can choose an item to replace the content of the edit box with. ***Science of Consciousness:*** "If a man is able to submit himself to nature, then nature will react to his needs automatically".

- Spinners provide a quick way to select one value from a set. In the default state, a spinner shows its currently selected value. Touching the spinner displays a dropdown menu with all other available values, from which the user can select a new one. ListView is a ViewGroup that creates a list of scrollable items. The list items are automatically inserted to the list using a ListAdapter, and listener find out when the selection has made. ***Science of Consciousness:*** *"Similarly every person having their own list of aims and choices in their life. Transcendental meditation is works like Android event listener what one wants to do in a better way, in a right way, for maximum results."*

Example :Recipe App using Explicit Intents and Customized GridView
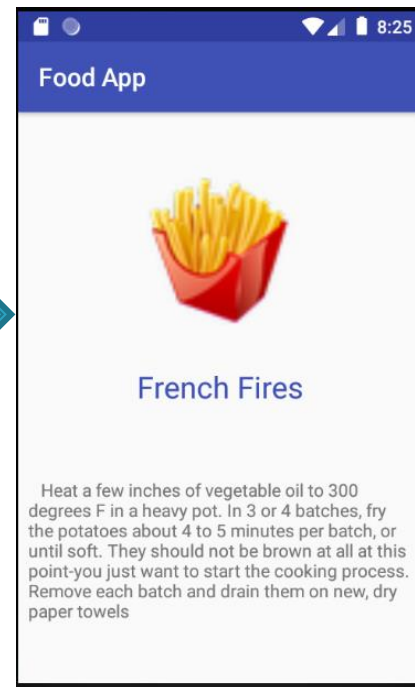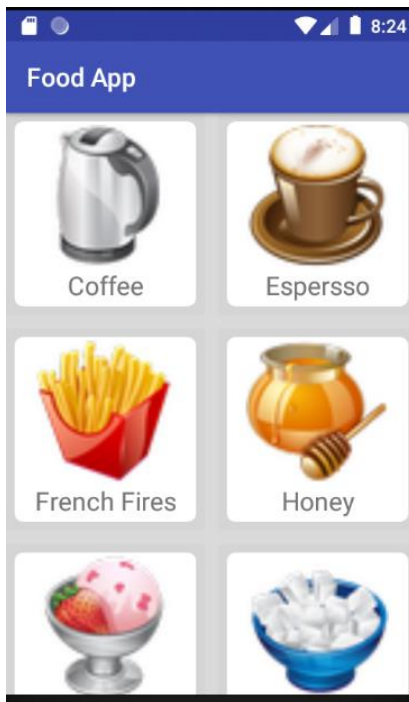
Example :Recipe App using Explicit Intents and Customized GridView classes and layouts

- MainAcvtivity.kt → activity_main.xml
- FoodDetails.kt → activity_food_details.xml
- Food.kt → Input type class / Data class
- food_ticket.xml → View to display each item in the GridView as like with its Image and its title below.

- To customize view create an inner class which inherit from BaseAdapter and override its methods.

More Detail refer : ReceipeApp

Android Dialogs

- Dialogs are prompt or alert displayed to the user to take a decision or to input any information. The dialogs are also used to notify user when a task has been completed. It does not fill the entire screen and usually appears when a user must take a particular action before proceeding.
- Android supports different types of Dialogs
  - Alert Dialog
  - Date Picker
  - Time Picker
  - Custom Dialog
  - Progress Dialog
  - Dialog Fragment

Alert Dialogs Example

- Alert Dialog is one of the built-in Dialog box with few functionalities like title, message and icon.
- We can create three possible choices of buttons (setPositiveButton(), setNegativeButton() and setNeutralButton().
  Refer Demo : AlertDemo.
- For more info https://developer.android.com/guide/topics/ui/dialogs.html
- Sample alert dialog

```
var builder = AlertDialog.Builder(this)
```

```
builder.setIcon(R.drawable.alerticon);          ⓘ Alert Message          builder.setTitle("Alert Message");

builder.setMessage("Do you want to Exit");       Do you want to Exit

builder.setNegativeButton("No",listener);       NO     YES      builder.setPositiveButton("Yes",listener);
```

```
val dialog: AlertDialog = builder.create()
dialog.show()
```

```kotlin
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        bt.setOnClickListener(object: View.OnClickListener {
            override  fun onClick(view:View) {
                // 1. Create an object for AlertDialog by passing the current context object
                val builder = AlertDialog.Builder( context: this@MainActivity)
                // 2. Set the basic information for the builder object
                builder.setTitle("Alert Message")
                builder.setMessage("Do you want to Exit")
                builder.setIcon(R.drawable.alerticon)
                // 3. Performing positive action on clicking Yes button
                builder.setPositiveButton( text: "Yes"){dialogInterface, which ->
                    dialogInterface.dismiss() // dismiss the dialog
                    finish() // to destroy the activity
                }
                // 4. Performing Cancel action on clicking Cancel button
                builder.setNegativeButton( text: "Cancel"){dialogInterface, which ->
                    dialogInterface.dismiss() // dismiss the dialog, but activity is still alive
                }
                // 5. Finally, make the alert dialog using builder
                val dialog: AlertDialog = builder.create()
                // 6. Display the alert dialog on app interface
                dialog.show()
            }
        })
    }
}
```
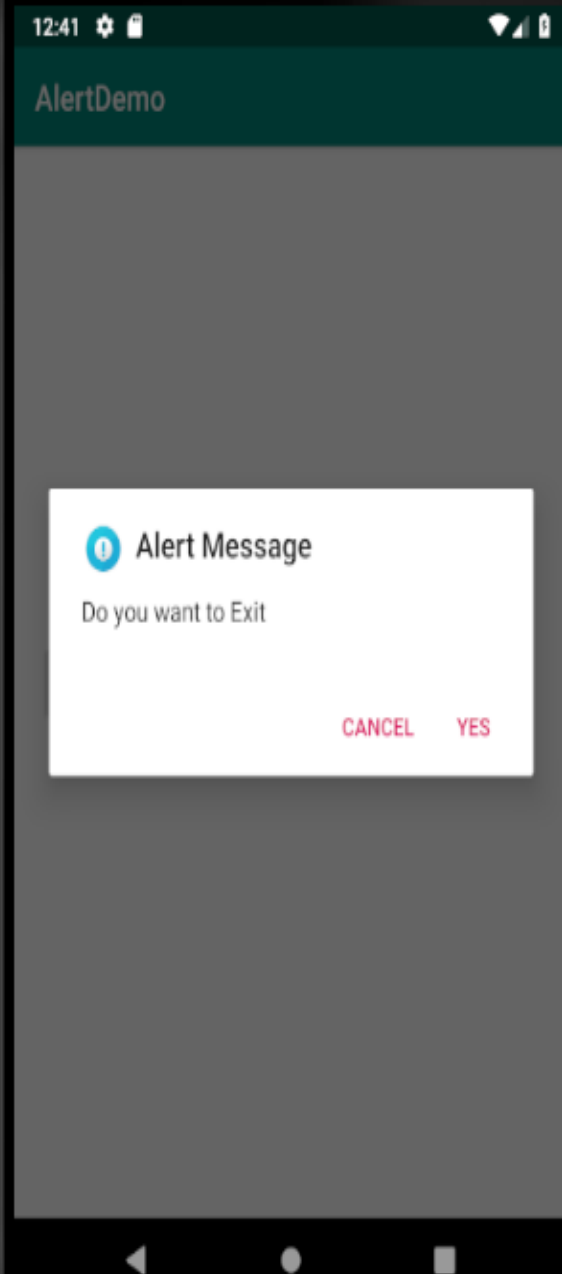
Date and Time Picker Dialogs

- **DatePickerDialog** and **TimePickerDialog** classes have onDateSetListener() and onTimeSetListener()callback methods, respectively.

- These callback methods are invoked when the user is done with selecting the date and time, respectively.

- The DatePickerDialog class consists of a 5-argument constructor with the parameters listed below.

  1. Context: It requires the application context

2. CallBack Function: onDateSetListener() is invoked and need to override the given method.

void onDateSet(DatePicker view, int year, int month, int dayOfMonth);

Parameters are

- view the picker associated with the dialog
- year the selected year
- month the selected month (0-11)
- dayOfMonth selected day of the month (1-31, depending on month)

3. int mYear : It shows the current year that's visible when the dialog pops up

4. int mMonth : It shows the current month that's visible when the dialog pops up

5. int mDay : It shows the current day that's visible when the dialog pops up

- The TimePickerDialog class consists of a 5-argument constructor with the parameters listed below.

  1. Context: It requires the application context

  2. CallBack Function: onTimeSetListener() is invoked when the user sets the time. Need to override the method

  void onTimeSet(TimePicker view, int hourOfDay, int minute);

  Parameter are
  - view the view associated with this listener
  - hourOfDay the hour that was set
  - minute the minute that was set

  3. int mHours : It shows the current Hour that's visible when the dialog pops up

  4. int mMinute : It shows the current minute that's visible when the dialog pops up

  5. boolean false : If its set to false it will show the time in 24 hour format else not

Example - Date and Time Picker Dialogs

- Problem Requirement :

  If you click the DATE

  PICKER button, it will open

  DatePicker dialog, chosen date

  will be replaced with Date

  TextView. Similar way for

  TIME PICKER.

- Refer : DateTimePickerDemo

Code for DatePicker Dialogs

```kotlin
// DatePicker Implementation
when(v?.id){
    R.id.bt1 -> {
        val c = Calendar.getInstance()
        val mYear = c.get(Calendar.YEAR)
        val  mMonth = c.get(Calendar.MONTH) +1
        val  mDay = c.get(Calendar.DAY_OF_MONTH)
        val dpd = DatePickerDialog(this,
        DatePickerDialog.OnDateSetListener { view, year, monthOfYear,
        dayOfMonth ->
            // Display Selected date in Toast
            tv1.text = "$dayOfMonth  $monthOfYear  $year"

        }, mYear, mMonth, mDay)

         dpd.show()
    }
```

```
 //  TimePicker Implementation
R.id.bt2 -> {
        val cal = Calendar.getInstance()
        val timeSetListener =
         TimePickerDialog.OnTimeSetListener { timePicker, hour,
         minute ->
            cal.set(Calendar.HOUR_OF_DAY, hour)
            cal.set(Calendar.MINUTE, minute)
            tv2.text = SimpleDateFormat("HH:mm").format(cal.time)
        }
        TimePickerDialog(this, timeSetListener,
cal.get(Calendar.HOUR_OF_DAY), cal.get(Calendar.MINUTE),
false).show()

       }
```

Main Point 3

- A dialog is a small window that prompts the user to decide or enter additional information. Customization of views and dialogs helps to show UI's in a better way. ***Science of Consciousness:*** *TM is a common interface to perform greater activities and taking right decision in life. Also helps to customize the individual and society needs in a better way.*