UNIT II

CHAPTER 2

User Experience

Syllabus

University Prescribed Syllabus

User Input Controls, Menus, Screen Navigation, RecyclerView

Syllabus Topic : User Input Controls

2.1 User Input Controls

- **Input controls** are the interactive components in app's **user** interface. To provide some function to a user, and in order to use it, there must be a way for the user to interact with App. i.e. **Android** provides different types of **controls** you can **use** in your **User** Interface (UI)
- For an Android app, includes tapping, pressing, typing, or talking and listening. And the framework provides corresponding user interface (UI) elements such as buttons, menus, keyboards, text entry fields, seek bars, checkboxes, zoom buttons, toggle buttons radio buttons, spinners, and more.
- User **input controls**, which are the interactive components in your app's user interface. You can use a wide variety of input controls in your UI.

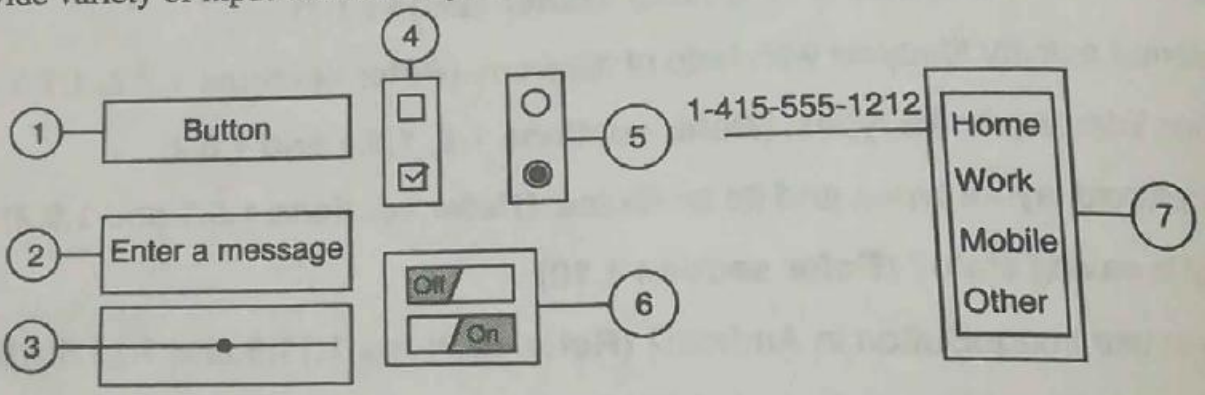


Fig. 2.1.1

- In the Fig. 2.1.1
  1. Button
  2. Text field
  3. Seek bar
  4. Checkboxes
  5. Radio buttons
  6. Toggle
  7. Spinner

Common Controls

Control	Type-Description-Related	Classes
Button	A push-button that can be pressed, or clicked, by the user to perform an action.	Button
Text field	An editable text field. You can use the AutoCompleteTextView widget to create a text entry widget that provides auto-complete suggestions- EditText, AutoCompleteTextView	
Checkbox	An on/off switch that can be toggled by the user. You should use checkboxes when presenting users with a group of selectable options that are not mutually exclusive.	CheckBox
Radio button	Similar to checkboxes, except that only one option can be selected in the group.	RadioGroup, RadioButton
Toggle button	An on/off button with a light indicator.	ToggleButton
Spinner	A drop-down list that allows users to select one value from a set.	Spinner
Pickers	A dialog for users to select a single value for a set by using up/down buttons or via a swipe gesture. Use a DatePickercode> widget to enter the values for the date (month, day, year) or a TimePickerwidget to enter the values for a time (hour, minute, AM/PM), which will be formatted automatically for the user's locale.	DatePicker, TimePicker

- Android apply a common programmatic abstraction to all input controls called a view. The View class represents the basic building block for UI components, including input controls.
- View is the base class for classes that provide support for interactive UI components, such as buttons, text fields, and layout managers.
- If there are many UI input components in your app, which one gets input from the user first? Consider in app there are several TextView objects and an EditText object then , which UI component (that is, which View) receives text typed by the user first?
- Hence to receives text typed by user first ,View required "has the focus" component that receives user input.

Attribute of an input control

1. Focusable

- focusable means that the view is allowed to gain focus from an input device such as a keyboard.
- Focus : Means which view is currently selected to receive input.
- It can be initiated by the user by touching a View, such as a TextView or an EditText object.
- Focus can also be programmatically controlled; a programmer can requestFocus() on any View that is focusable.
- Input devices like keyboards can't determine which view to send their input events to, so they send them to the view that has focus.

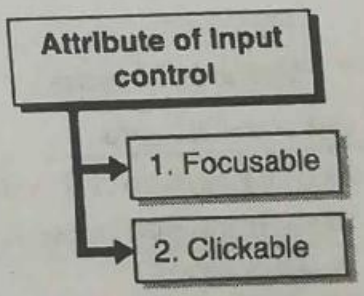


Fig. C.2.1

- When the user begins interacting with the interface by touching it, only Views with isFocusableInTouchMode() set to true are focusable, such as text input fields.
- Other Views that are touchable, such as buttons, do not take focus when touched.
- Focus movement is based on an algorithm that finds the nearest neighbour in a given direction:
- If you set an EditText view to a single-line, the user can tap the Return key on the keyboard to close the keyboard
- When the user touches the screen, the topmost view under the touch is in focus
- The system usually finds the nearest input control in the same direction the user was navigating (up, down, left, or right). And if multiple input controls that are nearby and in the same direction, the system scans from left to right, top to bottom.

➜ **2. Clickable**

- Clickable means the view can be clicked or tapped .
- If this attribute is (boolean) true, then the View can react to click events. As it is with focus, clickable can be programmatically controlled.
- You can override it by adding , if algorithm does not give you hence dd one of these attributes to a view to decide where to go upon leaving the view—in other words, which view should be the next view.
- And define the value of the attribute to be the id of the next view nextFocusDown, nextFocusLeft, nextFocusRight, and nextFocusUp XML attributes to your layout file.

```
<LinearLayout
android:orientation="vertical" >
<Button android:id="@+id/top"
android:nextFocusUp="@+id/bottom" />
<Button android:id="@+id/bottom"
android:nextFocusDown="@+id/top" />
</LinearLayout>
```

- If you declare a View as focusable in your, add the android:focusable XML attribute to the View in the layout, and set its value to true.
- Or declare a View as focusable while in "touch mode" with android:focusableInTouchMode set to true.

☞ **Methods of focus**

onFocusChanged	determine where focus came from.
Activity.getCurrentFocus(),or ViewGroup.getFocusedChild()	find out which view currently has the focus
findFocus()	find the view in the hierarchy that currently has focus
requestFocus()	give focus to a specific view
setFocusable()	change whether a view can take focus
setOnFocusChangeListener()	set a listener that will be notified when the view gains or loses focus.

2.1.1 Buttons

Fig. 2.1.2

- When touched or clicked, a button performs an action. The text and/or icon provide a hint of that action. It is also referred to as a "push-button" in Android documentation.
- A button is a rectangle or rounded rectangle, wider than it is tall, with a descriptive caption in its center.
- Android has several types of buttons, like raised buttons and flat buttons with three states: normal, disabled, and pressed.

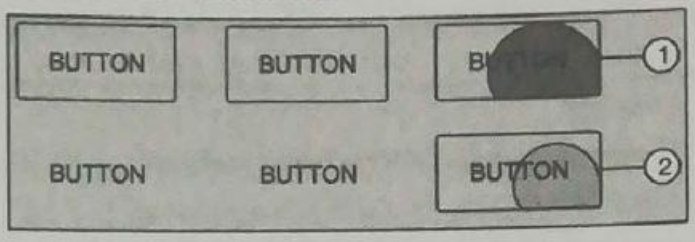


Fig. 2.1.3

- In the Fig. 2.1.3.
  1. Raised button in three states: normal, disabled, and pressed.
  2. Flat button in three states: normal, disabled, and pressed.
- Raised buttons add dimension to a flat layout : They emphasize functions on busy or wide spaces. Raised buttons show a background shadow when touched (pressed) or clicked, as shown in Fig. 2.1.4.

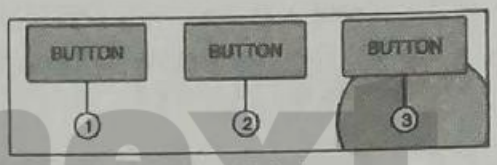


Fig. 2.1.4

- In the Fig. 2.1.4
  1. **Normal state :** The button looks like a raised button.
  2. **Disabled state :** When the button is disabled, it is grayed out and it's not active.
  3. **Pressed state :** It is with a larger background shadow, the button is being touched or clicked. When you attach a callback to the button (OnClick attribute), the callback is called when the button is in this state.
- The raised button can show text by using text attribute

```
<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
android:text="@string/button_text"
/>
```

- The raised button can show **with text and icon with text and** drawableLeft attribute
- To create a button with text and an icon as shown in the figure below, use a Button in your XML layout. Add the android:drawableLeft attribute to draw the icon to the left of the button's text, as shown in the figure below:

```
<Button
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
android:text="@string/button_text"
android:drawableLeft="@drawable/button_icon"
/>
```

- The raised button can show image (no text), use src attribute
- To display on image in raised button use the ImageButton class, which extends the ImageView class. You can add an ImageButton to your XML layout as follows:

```
<ImageButton
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:src="@drawable/button_icon" />
```

- The raised button can Chang the style and appearance by using a different background color for the button with android:background attribute with a drawable or color resource

```
<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
android:background="@color/colorPrimary"
/>
```

☞ **Adding the button with text and icon to the layout**

To create a button with text and an icon as shown in the figure below, use a Button in your XML layout. Add the android:drawableLeft attribute to draw the icon to the left of the button's text, as shown in the Fig. 2.1.5.

2.1.2 Designing Flat Buttons

- A flat button, also called as borderless button, It is a text-only button and appears flat on the screen without a shadow. It is simplicity and useful when you have a dialog, as shown in the figure below, which requires user input or interaction.

Fig. 2.1.5

- If, you would want to have the same font and style as the text surrounding the button. This keeps the look and feel the same across all elements within the dialog.
- Flat buttons are resemble basic buttons except that they have no borders or background, but still change appearance during different states. It shows an ink shade around it when pressed (touched or clicked).

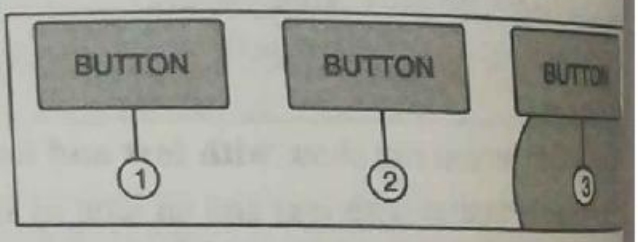


Fig. 2.1.6

- In the Fig. 2.1.6.
  1. **Normal state :** the button looks like a ordinary text.
  2. **Disabled state :** When the button is disabled, it is grayed out and it's not active.
  3. **Pressed state :** It is with a background shadow, the button is being touched or clicked. When you attach a callback to the button (OnClick attribute), the callback is called when the button is in this state.
- **To** use a flat button within a layout, use padding to set it off from the surrounding text, so that user can easily see it.

- For creating a flat button, use the Button class. And Add a Button in XML layout, and apply style attribute as "?android:attr/borderlessButtonStyle"

```
<Button
android:id="@+id/button_send"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/button_send"
android:onClick="sendMessage"
style="?android:attr/borderlessButtonStyle" />
```

2.1.3 Designing Images as Buttons

- You can design button with ImageView, by adding the android:onClick attribute in the XML layout.
- The image for the ImageView must already be stored in the **drawables** folder of your project.
- To bring images into your Android Studio project, download required image and save the image in JPEG format, and then copy the image file into the app > src > main > res > drawables folder of your project.
- As shown in Fig. 2.1.7.
- If you want to use multiple images as buttons, arrange them in a viewgroup so that all required images are grouped together.
- For example : The following images in the drawable folder (icecream_circle.jpg, donut_circle.jpg, and froyo_circle.jpg) are defined for ImageViews that are grouped in a LinearLayout set to a horizontal orientation so that they appear side-by-side.

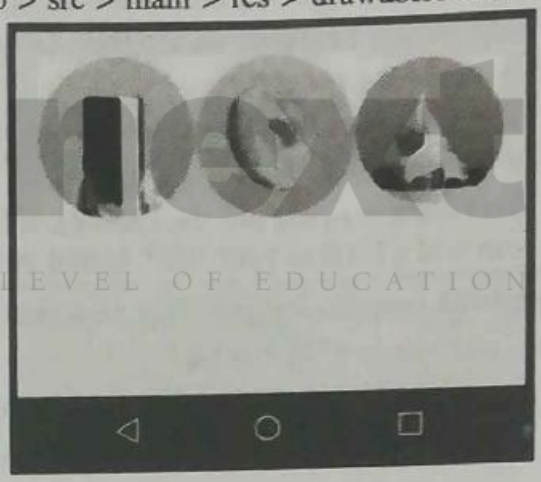


Fig. 2.1.7

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    android:layout_marginTop="260dp">
    <ImageView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:src="@drawable/icecream_circle"
      android:onClick="orderIcecream"/>
    <ImageView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:src="@drawable/donut_circle"
```

```
        android:onClick="orderDonut"/>
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/froyo_circle"
        android:onClick="orderFroyo"/>
</LinearLayout>
```

2.1.4 Floating Action Button (FAB)

- A floating action button, in Fig. 2.1.8, is a circular button that appears to float above the layout.
- FAB used to only represent the primary action for a screen.
- For example, the primary action for what's app app's call screen is select a contact for calling purpose, as shown in the figure above.
- FAB is the right choice action to be persistent and readily available on a screen. Only one FAB is suggested per screen.
- The FAB uses the same type of icons that use for a button with an icon.

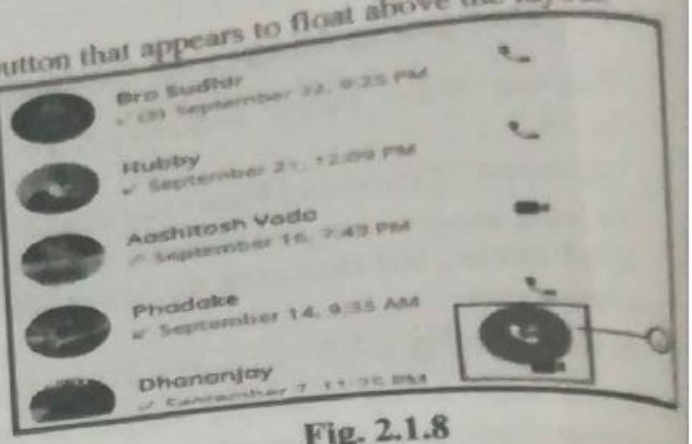


Fig. 2.1.8

- To create a FAB, use the FloatingActionButton class, which extends the ImageButton class. You can add a FAB to your XML layout as follows:

```
<android.support.design.widget.FloatingActionButton
    android:id="@+id/fab"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom|end"
    android:layout_margin="@dimen/fab_margin"
    android:src="@drawable/ic_fab_chat_button_white" />
```

- FAB, by default, are 56 x 56 dp in size. It is best to use the default size unless you need the smaller version to create visual continuity with other screen elements.
- You can set the mini size (30 x 40 dp) with the app:fabSize attribute:

```
app:fabSize="mini"
```

- To set it back to the default size (56 x 56 dp):

```
app:fabSize="normal"
```

2.1.5 Responding to Button-Click Events

- When the user taps or clicks a clickable object, such as a Button, ImageButton, FloatingActionButton. Use an event listener called OnClickListener, which is an interface the View class, to respond to the click event that occurs on particular button .

Adding onClick to the layout element

- Add OnClickListener for the clickable object in your Activity code and must be necessary to add a callback method, with attribute as android:onClick with the clickable object's element in the XML layout.

Example

```
<Button
android:id="@+id/button_send"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/button_send"
android:onClick="sendMessage" />
```

- Here when a user clicks the button, the Android system calls the Activity's sendMessage() method:
- This method is write in java file as follows.

```
public void sendMessage(View view) {
  // Do something in response to button click
}
```

- When the method you declare as name here sendMessage must be public, return void, and define a View as its only parameter. So use the method to perform a task or call other methods as a response to the button click.
- Using the button-listener design pattern
- You can also handle the click event programmatically using the button-listener design pattern (see Fig. 2.1.9).

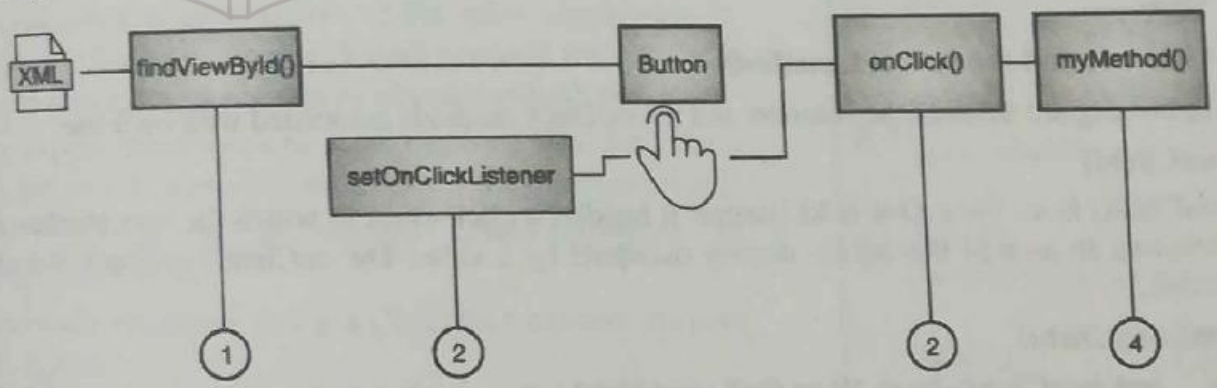


Fig. 2.1.9

- Use the event listener name as View.OnClickListener, which is an interface in the View class and it contains a single callback method, name as onClick().when the view is triggered by user interaction this method is called by the Android framework
- The event listener must already be registered to the view in order to be called for the event. As given below
- Use the findViewById() method of the View class to find the button in the XML layout file:

```
Button button =(Button)findViewById(R.id.button_send);
        ⇩                ⇩          ⇩  ⇩  ⇩
    Objectname       mehodname resource id nameofid
```

```
Button button =(Button)findViewById(R.id.button_send);
```

- Get a new View.OnClickListener object and register it to the button by calling the setOnClickListener() method. and argument to setOnClickListener() takes an object which implements the View.OnClickListener interface, which has method as : onClick().

```
button.setOnClickListener(new View.OnClickListener()
{ public void onClick(View v) {
   // Do something in response to button click
}
```

- Here define the method onClick() to be public,return void, and define a View as its only parameter.

☞ **Example**

```
fab.setOnClickListener(new View.OnClickListener()
{       @Override
 public void onClick(View view) {
// Add a new word to the wordList.
        }
});
```

- You can also use the callback methods already defined in the event listener interfaces to handle them.

☞ **Listeners and the callback methods**

- Following are some of the listeners and the callback methods associated with each one.

1. **onClick()**
   onClick() from View.OnClickListener: it handles a click event in which the user touches and then releases an area of the device display occupied by a view. The onClick() callback has no return value.
2. **onLongClick()**
   - **onLongClick() from View.OnLongClickListener :** It handles an event in which the user maintains the touch over a view for an extended period. This returns a boolean to indicate whether you have consumed the event and it should not be carried further.
   - It return true you have handled the event and it should stop here
   - It return false if you have not handled it and/or the event should continue to any other on-click listeners.
3. **onTouch()**
   - **onTouch() from View.OnTouchListener :** It handles any form of touch contact with the screen including individual or multiple touches and gesture motions, including a press, release, or any movement gesture on the screen.

- A MotionEvent is passed as an argument, which includes directional information, and it returns a boolean to indicate whether your listener consumes this event.

4. **onFocusChange()**

onFocusChange() from View.OnFocusChangeListener: Handles when focus moves away from the current view as the result of interaction with a trackball or navigation key.

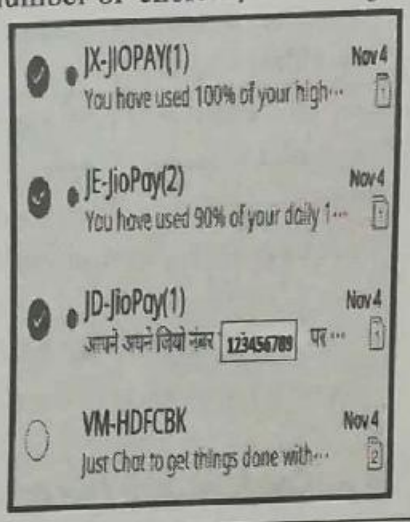
5. **onKey()**

- onKey() from View.OnKeyListener: Handles when a key on a hardware device is pressed while a view has focus.
- **For making choices following are** ready-made input controls for the user to select one or more choices.

Name	Use
Checkboxes	Select one or more values from a set of values by clicking each value's checkbox
Radio buttons	Select only one value from a set of values by clicking the value's circular "radio" button. If you are providing only two or three choices, you might want to use radio buttons for the choices if you have room in your layout for them.
Toggle button	Select one state out of two or more states. Toggle buttons usually offer two visible states, such as "on" and "off".
Spinner	Select one value from a set of values in a drop-down menu. Only one value can be selected

2.1.6 Checkboxes

- When you have a list of options and the user may select any number of choices, including no choices.
- Each checkbox is independent of the other checkboxes in the list, so checking one box doesn't uncheck the others. A user can also uncheck an already checked checkbox.
- Users expect checkboxes to appear in a vertical list, like a to-do list, or side-by-side horizontally if the labels are short.
- Each checkbox is a separate instance(object) of the CheckBox class.
- Create each checkbox using a CheckBox element in your XML layout.
- To create multiple checkboxes in a vertical orientation, use a vertical LinearLayout: as given below.




```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <CheckBox android:id="@+id/checkbox1_chocolate"
      android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:text="@string/chocolate_syrup" />
    <CheckBox android:id="@+id/checkbox2_sprinkles"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/sprinkles" />
    <CheckBox android:id="@+id/checkbox3_nuts"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/crushed_nuts" />
</LinearLayout>
```

- For retrieve the state of checkboxes when a user touches or clicks a **Submit** or **Done** button in the same activity, which uses the android:onClick attribute to call a method such as onSubmit():

```
<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/submit"
  android:onClick="onSubmit"/>
```

- If a checkbox is selected by using the isChecked() method
- The isChecked() method will return a (boolean) true if there is a checkmark in the box.
- The boolean value of true or false to checked depending on whether the checkbox is checked :

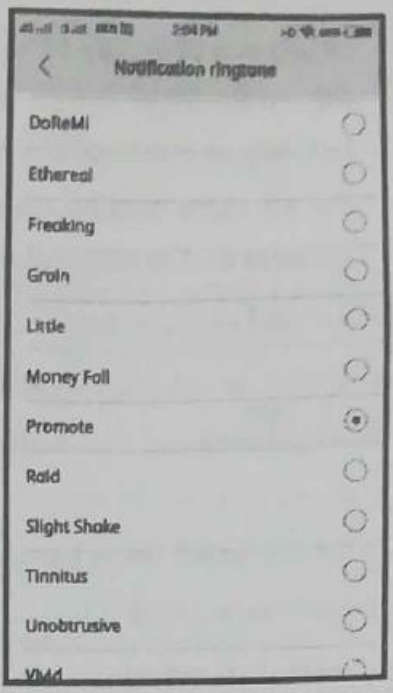
```
boolean checked = ((CheckBox) view).isChecked();
```

- consider onSubmit() method check to see which checkbox is selected, using the resource id for the checkbox element:

```
public void onSubmit(View view){
StringBuffer toppings = new StringBuffer().append(getString(R.string.toppings_label));
  if (((CheckBox) findViewById(R.id.checkbox1_chocolate)).isChecked()) {
    toppings.append(getString(R.string.chocolate_syrup_text));
  }
  if (((CheckBox) findViewById(R.id.checkbox2_sprinkles)).isChecked()) {
    toppings.append(getString(R.string.sprinkles_text));
  }
  if (((CheckBox) findViewById(R.id.checkbox3_nuts)).isChecked()) {
    toppings.append(getString(R.string.crushed_nuts_text));
  }
}
```

- You can use the android:onClick attribute in the XML layout for each checkbox to declare the callback method for that checkbox, which must be defined within the activity that hosts this layout.

2.1.7 Radio Buttons



- Use radio buttons when you have two or more options that are mutually exclusive the user must select only one of them.
- Each radio button is an instance of the RadioButton class. Radio buttons are normally used together in a RadioGroup.
- When several radio buttons live inside a radio group, checking one radio button unchecks all the others.
- You create each radio button using a RadioButton element in your XML layout within a RadioGroup view group:

```
<RadioGroup
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:layout_below="@id/orderintrotext">
    <RadioButton
        android:id="@+id/sameday"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/same_day_messenger_service"
        android:onClick="onRadioButtonClicked"/>
    <RadioButton
        android:id="@+id/nextday"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/next_day_ground_delivery"
        android:onClick="onRadioButtonClicked"/>
    <RadioButton
        android:id="@+id/pickup"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/pick_up"
        android:onClick="onRadioButtonClicked"/>
</RadioGroup>
```

- Use the android:onClick attribute for each radio button to declare the click event handler method for the radio button, which must be defined within the activity that hosts this layout.

- When you click any radio button then it calls same onRadioButtonClicked() method in the activity, but you could create separate methods in the activity and declare them in each radio button's android:onClick attribute.
- for all radio buttons, using switch case statements to check the resource id for the radio button element to determine which one was checked:

```
public void onRadioButtonClicked(View view) {
  // Check to see if a button has been clicked.
  boolean checked = ((RadioButton) view).isChecked();

  // Check which radio button was clicked.
  switch(view.getId()) {
    case R.id.sameday:
      if (checked)
        // Same day service
        break;
    case R.id.nextday:
      if (checked)
        // Next day delivery
        break;
    case R.id.pickup:
      if (checked)
        // Pick up
        break;
  }
}
```


- The android:onClick attributes from the radio buttons. Then add the onRadioButtonClicked method to the android:onClick attribute for the **Submit** or **Done** button.

2.1.8 Toggle Buttons

- A toggle input control lets the user change a setting between two states. Android provides a ToggleButton class, which shows a raised button with "OFF" and "ON".
- Toggles include the On/Off switches for Wi-Fi, Bluetooth, and other options in the Settings app.
- Android also provides the Switch class, which is a short slider that looks like a rocker switch offering two states (on and off). Both are extensions of the CompoundButton class.

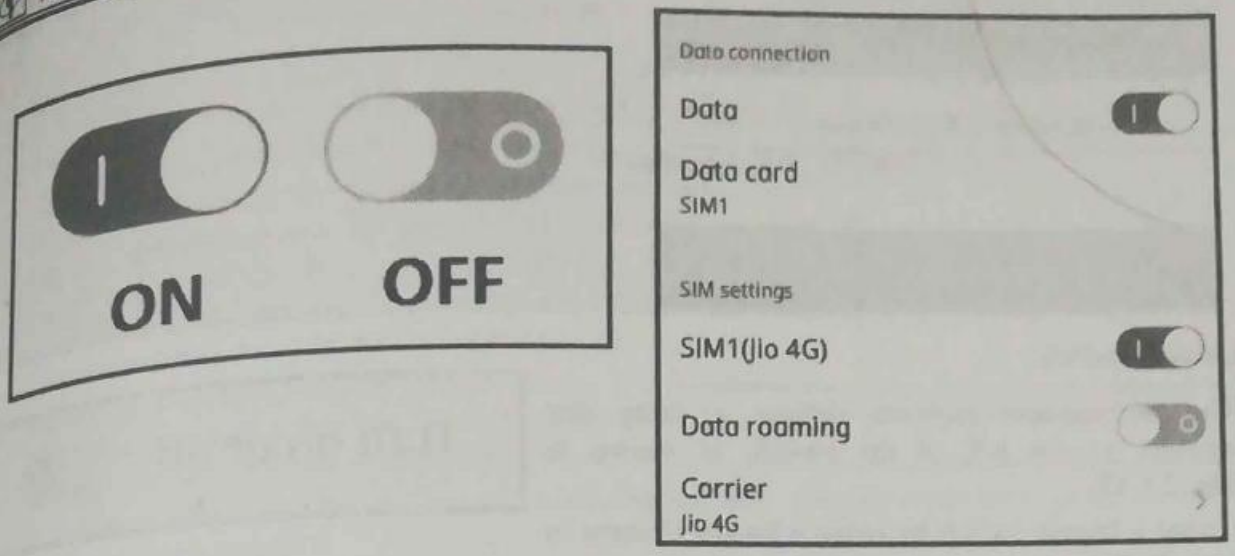


Fig. 2.1.10

☞ **Using a toggle button**

Create a toggle button by using a ToggleButton element in your XML layout:

```
<ToggleButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/my_toggle"
    android:text="
    android:onClick="onToggleClick"/>
```

- Toggle button always shows either "ON" or "OFF".
- So android:text attribute does not provide a text label for a toggle button
- To provide a text label next to (or above) the toggle button, use a separate TextView.
- To respond to the toggle tap, declare an android:onClick callback method for the ToggleButton.
- Use CompoundButton.OnCheckedChangeListener() to detect the state change of the toggle. Create a CompoundButton.OnCheckedChangeListener object and assign it to the button by calling setOnCheckedChangeListener().
- The onToggleClick() method checks whether the toggle is on or off, and displays a toast message:

```
public void onToggleClick(View view) {
  ToggleButton toggle = (ToggleButton) findViewById(R.id.my_toggle);
  toggle.setOnCheckedChangeListener(new
                    CompoundButton.OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButton buttonView,
                    boolean isChecked) {
      StringBuffer onOff = new StringBuffer().append("On or off? ");
      if (isChecked) { // The toggle is enabled
        onOff.append("ON ");
```

```
    } else { // The toggle is disabled
      onOff.append("OFF ");
    }
    Toast.makeText(getApplicationContext(), onOff.toString(),
                Toast.LENGTH_SHORT).show();

    }
  });
}
```

Using a switch

- The android:text attribute defines a string that appears to the left of the switch, as shown in Fig. 2.1.11.
- Create a toggle switch by using a Switch element in your XML layout.

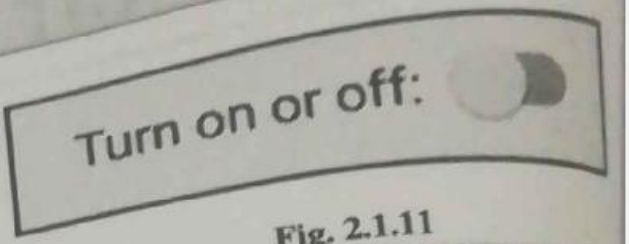
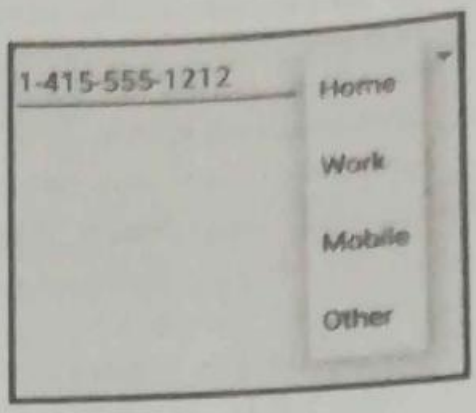


Fig. 2.1.11

```
<Switch
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/my_switch"
    android:text="@string/turn_on_or_off"
    android:onClick="onSwitchClick"/>

public void onSwitchClick(View view) {
  Switch aSwitch = (Switch) findViewById(R.id.my_switch);
  aSwitch.setOnCheckedChangeListener(new
              CompoundButton.OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButton buttonView,
              boolean isChecked) {
  StringBuffer onOff = new StringBuffer().append("On or off? ");
    if (isChecked) { // The switch is enabled
      onOff.append("ON ");
    } else { // The switch is disabled
      onOff.append("OFF ");
    }
    Toast.makeText(getApplicationContext(), onOff.toString(),
              Toast.LENGTH_SHORT).show();

  }
 });
}
```

2.2 Spinners



- A spinner provides a quick way to select one value from a set. Touching the spinner displays a drop-down list with all available values, from which the user can select one.
- If you have a long list of choices, a spinner may extend beyond your layout, forcing the user to scroll it. A spinner scrolls automatically, with no extra code needed. However, scrolling a long list (such as a list of countries) is not recommended as it can be hard to select an item.
- To create a spinner, use the Spinner class, which creates a view that displays individual spinner values as child views, and lets the user pick one. Follow these steps:

1. Create a Spinner element in your XML layout, and specify its values using an array and an ArrayAdapter.
2. Create the spinner and its adapter using the SpinnerAdapter class.
3. To define the selection callback for the spinner, update the Activity that uses the spinner to implement the AdapterView.OnItemSelectedListener interface.

Create the spinner UI element

- To create a spinner in your XML layout, add a Spinner element, which provides the drop-down list:

```
<Spinner
android:id="@+id/label_spinner"
android:layout_width="wrap_content"
android:layout_height="wrap_content">
</Spinner>
```

Specify the values for the spinner

- You add an adapter that fills the spinner list with values. An adapter is like a bridge, or intermediary, between two incompatible interfaces.
- For example, a memory card reader acts as an adapter between the memory card and a laptop.
- You plug the memory card into the card reader, and plug the card reader into the laptop, so that the laptop can read the memory card.
- The spinner-adapter pattern takes the data set you've specified and makes a view for each item in the data set, as shown in the Fig. 2.2.1.

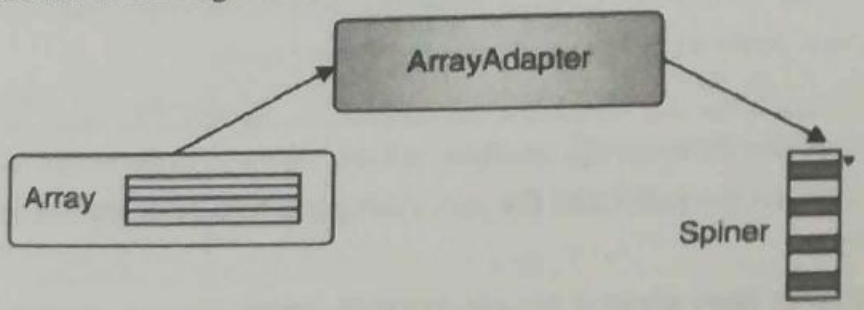


Fig. 2.2.1

- The SpinnerAdapter class, implements the Adapter class, allows you to define two different views:

Shows the data values in the spinner itself	Shows the data in the drop-down list when the spinner is touched or clicked.

- The values provide for the spinner can come from any source, but must be provided through a SpinnerAdapter, such as an ArrayAdapter if the values are available in an array.
- It contain a simple array called labels_array of predetermined values in the strings.xml file:

```
<string-array name="labels_array">
    <item>Home</item>
    <item>Work</item>
    <item>Mobile</item>
    <item>Other</item>
</string-array>
```

☞ **Create the spinner and its adapter**

- Create the spinner, and set its listener to the activity that implements the callback methods onCreate() method.
- Add the code below to the onCreate() method
- Gets the spinner object added to the layout using findViewById() to find it by its id(label_spinner).
- Sets the onItemSelectedListener to whichever activity implements the callbacks (this) using the setOnItemSelectedListener() method.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  // Create the spinner.
  Spinner spinner = (Spinner) findViewById(R.id.label_spinner);
  if (spinner != null) {
    spinner.setOnItemSelectedListener(this);
  }
}
```

- Also in the onCreate() method, add a statement that creates the ArrayAdapter with the string array

```
// Create ArrayAdapter using the string array and default spinner layout.
ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this,
        R.array.labels_array, android.R.layout.simple_spinner_item);
```

- Now you use the createFromResource() method, which takes as arguments:
- The activity that implements the callbacks for processing the results of the spinner (this)
- The array (labels_array)
- The layout for each spinner item (layout.simple_spinner_item).

- You should use the simple_spinner_item default layout, unless you want to define your own layout for the items in the spinner.
- Specify the layout the adapter should use to display the list of spinner choices by calling the setDropDownViewResource() method of the ArrayAdapter class.
- For example : simple_spinner_dropdown_item as your layout:

```
// Specify the layout to use when the list of choices appears.
adapter.setDropDownViewResource
        (android.R.layout.simple_spinner_dropdown_item);
```

- You should use the simple_spinner_dropdown_item default layout, unless you want to define your own layout for the spinner's appearance.
- Use setAdapter() to apply the adapter to the spinner:

```
// Apply the adapter to the spinner.
spinner.setAdapter(adapter);
```

The full code for the onCreate() method is shown below:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);

// Create the spinner.
Spinner spinner = (Spinner) findViewById(R.id.label_spinner);
if (spinner != null) {
    spinner.setOnItemSelectedListener(this);
}

// Create ArrayAdapter using the string array and default spinner layout.
ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this,
        R.array.labels_array, android.R.layout.simple_spinner_item);

// Specify the layout to use when the list of choices appears.
adapter.setDropDownViewResource
        (android.R.layout.simple_spinner_dropdown_item);

// Apply the adapter to the spinner.
if (spinner != null) {
    spinner.setAdapter(adapter);
} }
```

- Implement the OnItemSelectedListener interface in the Activity

- To define the selection callback for the spinner, update the Activity that uses the spinner to implement the AdapterView.OnItemSelectedListener interface:

```
public class MainActivity extends AppCompatActivity implements
        AdapterView.OnItemSelectedListener { }
```

- //Implement the AdapterView.OnItemSelectedListener interface in order to have the onItemSelected() and onNothingSelected() callback methods to use with the spinner object.
- Following steps are involved When the user chooses an item from the spinner's drop-down list
- The Spinner object receives an on-item-selected event.
  1. The event triggers the calling of the onItemSelected() callback method of the AdapterView.OnItemSelectedListener interface.
  2. Retrieve the selected item in the spinner menu using the getItemAtPosition() method of the AdapterView class:

```
public void onItemSelected(AdapterView<?> adapterView, View view, int pos, long id)
{
String spinner_item = adapterView.getItemAtPosition(pos).toString();
}
```

☞ **Arguments for onItemSelected()**

parent AdapterView	The AdapterView where the selection happened
View View	The view within the AdapterView that was clicked
int pos	The position of the view in the adapter
long id	The row id of the item that is selected

- Implement/override the onNothingSelected() callback method of the AdapterView.OnItemSelectedListener interface to do something if nothing is selected.

2.3 Text

- Use the EditText class to get user input that consists of textual characters, including numbers and symbols. EditText extends the TextView class, to make the TextView editable.

☞ **Customizing an EditText object for user input**

- Create an EditText view by adding an EditText to your layout with the following XML:

```
<EditText
  android:id="@+id/edit_simple"
  android:layout_height="wrap_content"
  android:layout_width="match_parent">
</EditText>
```

☞ **Enabling multiple lines of input**

- By default, the EditText view allows multiple lines of input as shown in the figure below and suggests spelling corrections also . Tapping the Enter) key on the on-screen keyboard enters and starts a new line in the same EditText view (Fig. 2.3.1)

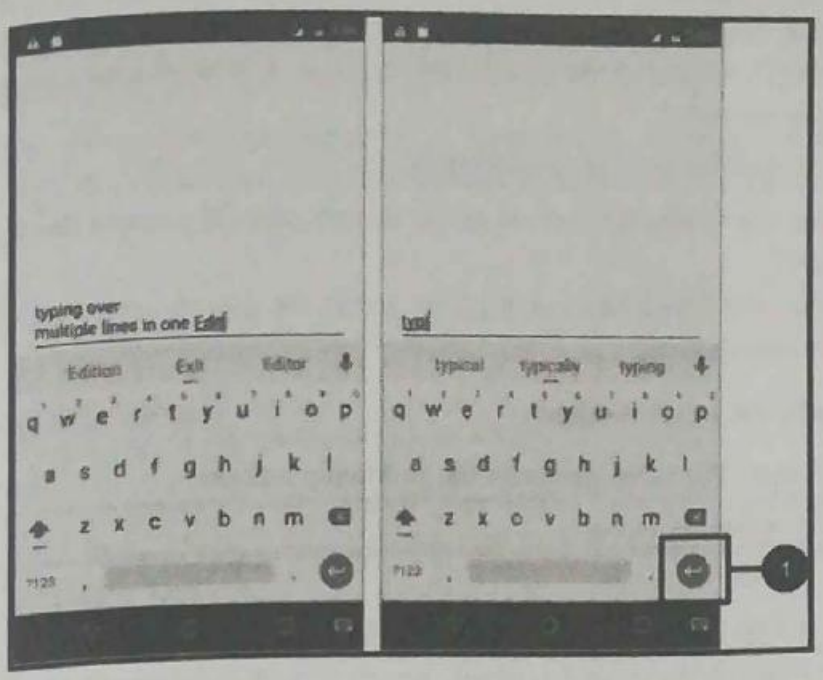


Fig. 2.3.1

Attributes for customizing an EditText view

- Use attributes to customize the EditText view for input.

Name of attribute	Use
android:maxLines="1"	Set the text entry to show only one line.
android:lines="2"	Set the text entry to show 2 lines, even if the length of the text is less
android:maxLength="5"	Set the maximum number of input characters to 5.
android:inputType="number"	Restrict text entry to numbers.
android:digits="01"	Restrict the digits entered to just "0" and "1".
android:textColorHighlight="#7cff88"	Set the background color of selected (highlighted) text.
android:hint="@string/my_hint":	Set text to appear in the field that provides a hint for the user, such as "Enter a message". Enter a message

- Create the EditText view element in the XML layout for an activity. Be sure to identify this element with an android:id so that you can refer to it by its id:

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

- In the Java code for the same activity, create a method with a View parameter that gets the EditText object (in the example below, editText) for the EditText view, using the findViewById()method of the View class to find the view by its id (editText_main):

```
EditText editText = (EditText) findViewById(R.id.editText_main);
```

- Use the getText() method of the EditText class (inherited from the TextView class) to obtain the text as a character sequence (CharSequence). You can convert the character sequence into a string using the toString() method of the CharSequence class, which returns a string representing the data in the character sequence.

```
String showString = editText.getText().toString();
```

- You can also use the valueOf() method of the Integer class to convert the string to an integer if the input is an integer.
- Android provide own Input Method Editors (IME) for speech input, specific types of keyboard entry, and other applications.

☞ **Attribute declare the input method**

- Use the android:inputType attribute with the following values.

android:inputType="attribute name "

Name of attribute	Use
phone	Sets the on-screen keyboard to be a phone keypad.
textCapSentence	Set the keyboard to capital letters at the beginning of a sentence.
textAutoCorrect	Enable spelling suggestions as the user types.
textPassword	Turn each character the user enters into a dot to conceal an entered password.
extEmailAddress	For email entry, show an email keyboard with the "@" symbol conveniently located next to the space key.

☞ **Example**

- The android:inputType attribute, in the above example, sets the keyboard type to phone, which forces one line of input (for a phone number).
- Use the android:inputType attribute to set an input type for the keyboard:

```
<EditText
  android:id="@+id/phone_number"
  android:inputType="phone"
  ... >
</EditText>
```

- Use setOnEditorActionListener() to set the listener for the EditText view to respond to the user the "action" key:

```
EditText editText = (EditText) findViewById(R.id.phone_number);
editText.setOnEditorActionListener(new
                    TextView.OnEditorActionListener() {
  // Add onEditorAction() method
}
```

- Use the IME_ACTION_SEND constant in the EditorInfo class for the actionId to show a Send as the "action" key, and create a method to respond to the pressed **Send** key (in case, dialNumberto dial the entered phone number):

```
@Override
public boolean onEditorAction(TextView textView,
```

```
                int actionId, KeyEvent keyEvent) {
boolean handled = false;
if (actionId == EditorInfo.IME_ACTION_SEND) {
dialNumber();
handled = true;
}
return handled;
});
```

2.4 Dialogs and Pickers

- A dialog is a window that appears on top of the display or fills the display, interrupting the flow of activity. Dialogs inform users about a specific task and may contain critical information, require decisions, or involve multiple tasks.
- Dialog use to show an alert that requires users to tap a button make a decision, such as **OK** or **Cancel**, **Disagree** and **Agree** buttons, and **Cancel** and **Discard** buttons.

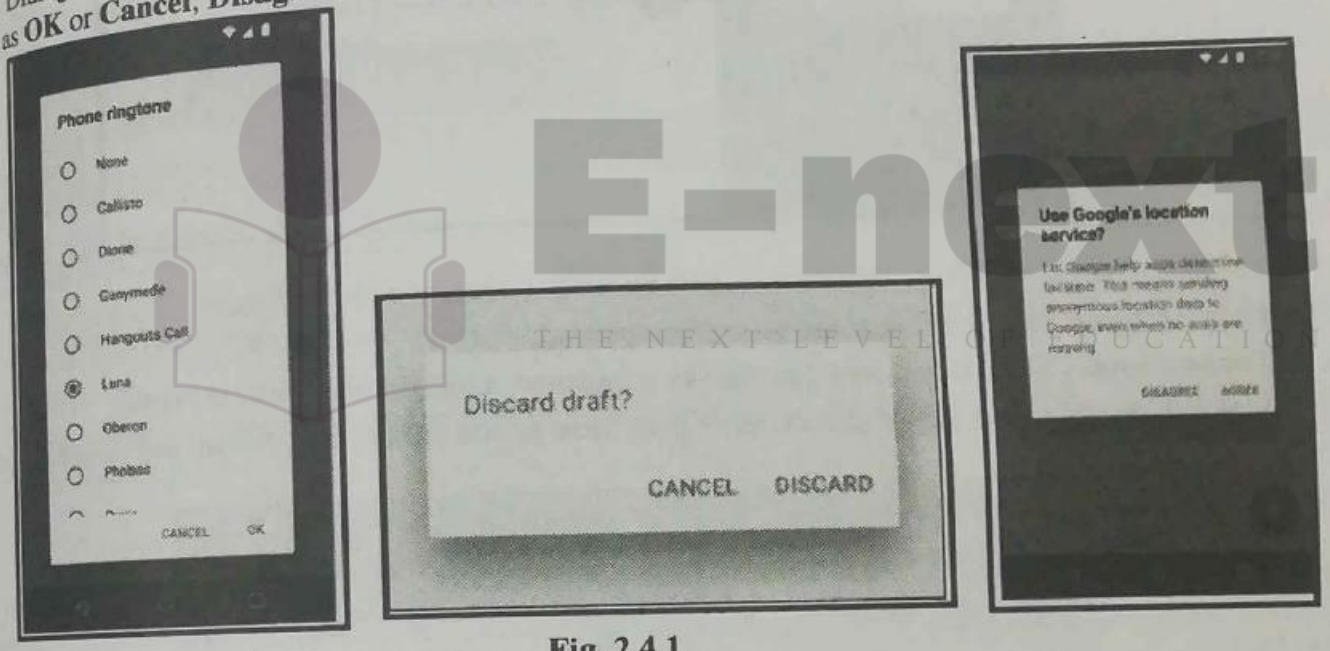


Fig. 2.4.1

- The base class for all dialog components is a Dialog.
- The Android SDK also provides ready-to-use dialog subclasses such as pickers for picking a time or a date, as shown on the right side of the figure below.
- Pickers allow users to enter information in a predetermined, consistent format that reduces the chance for input error.
- Dialogs always retain focus until dismissed or a required action has been taken.
- The Dialog class is the base class for dialogs, but you should avoid instantiating Dialog directly unless you are creating a custom dialog. For standard Android dialogs, use one of the following subclasses:

1. **AlertDialog** : A dialog that can show a title, up to three buttons, a list of selectable items, or a custom layout.
2. **DatePickerDialog** or **TimePickerDialog**: A dialog with a pre-defined UI that lets the user

select a date or time.

Showing an alert dialog

- Alerts are urgent interruptions, requiring acknowledgement or action, that informs the user about a situation as it occurs
- You can provide buttons in an alert to make a decision.
- Use the AlertDialog subclass of the Dialog class to show a standard dialog for an alert.

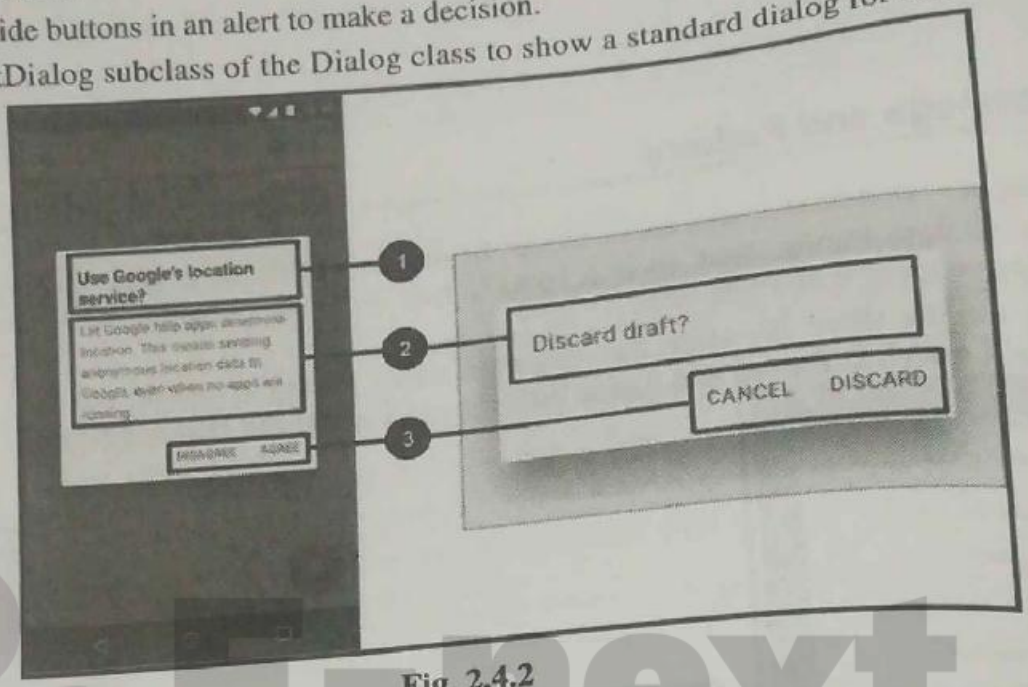


Fig. 2.4.2

1. **Title :** A title is title for dialog and it may be optional.
2. **Content area :** The content area can display a message, a list, or other custom layout.
3. **Action buttons :** You should use no more than three action buttons in a dialog, and most have only two.

- The AlertDialog.Builder class uses the builder design pattern.
- Use AlertDialog.Builder to build a standard alert dialog and set attributes on the dialog.

Name of attribute	Use
setTitle()	set its title
setMessage()	to set its message
setPositiveButton() and setNegativeButton()	set its buttons.

- The following creates the dialog object (myAlertBuilder) and sets the title and message (
- Here setTitle, resource called alert_title
- setMessage ,resource called alert_message

```
AlertDialog.Builder myAlertBuilder = new
        AlertDialog.Builder(MainActivity.this);
myAlertBuilder.setTitle(R.string.alert_title);
myAlertBuilder.setMessage(R.string.alert_message);
```

- Setting the button actions for the alert dialog

- Use the setPositiveButton() and setNegativeButton() methods of the AlertDialog.Builder class to set the button actions for the alert dialog. and the DialogInterface.OnClickListener class that defines the action to take when the user presses the button:

```
myAlertBuilder.setPositiveButton("OK", new DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialog, int which) {
    // User clicked OK button.

}
});
myAlertBuilder.setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialog, int which) {
    // User clicked the CANCEL button.

}
});
```

- You can also set a "neutral" button with setNeutralButton().
- Use a neutral button, such as "Remind me later", if you want the user to be able to dismiss the dialog and decide later.

Displaying the dialog

- To display the dialog, call its show() method:

```
alertDialog.show();
```

2.5 Date and Time Pickers

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

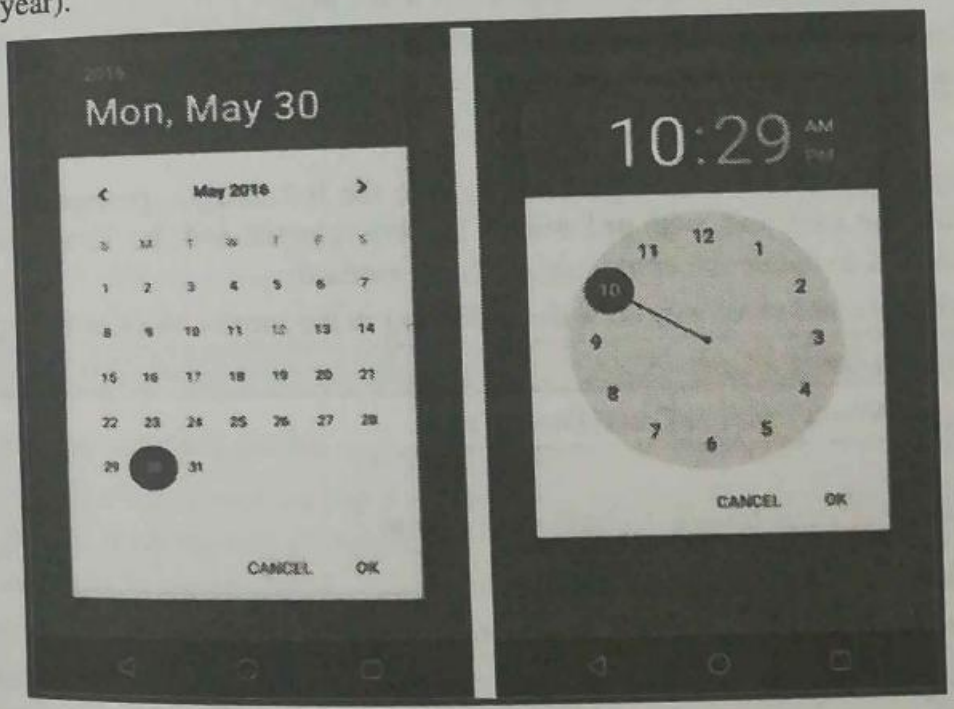


Fig. 2.5.1

- For a picker, you should use an instance of DialogFragment, a subclass of Fragment, which displays a dialog window floating on top of its activity's window.
- A fragment is a behavior or a portion of user interface within an activity. It's like a mini-activity within the main activity, with its own individual lifecycle.
- A fragment receives its own input events, and you can add or remove it while the main activity is running.
- You might combine multiple fragments in a single activity to build a multiple-pane user interface, or reuse a fragment in multiple activities.
- You can also use DialogFragment to manage the dialog lifecycle.

☞ **Adding a fragment**

- To add a fragment for the date picker, create a blank fragment (**DatePickerFragment**) without a layout XML, and without factory methods or interface callbacks:
- Expand **app > java > com.example.android.DateTimePickers** and select **MainActivity**.
- Choose **File > New > Fragment > Fragment (Blank)**, and name the fragment **DatePickerFragment**. Uncheck all three checkbox options so that you do not create a layout XML, do not include fragment factory methods, and do not include interface callbacks. You don't need to create a layout for a standard picker. Click **Finish** to create the fragment.

☞ **Extending DialogFragment for the picker**

- The next step is to create a standard picker with a listener. Follow these steps:
- Edit the DatePickerFragment class definition to extend DialogFragment, and implement DatePickerDialog.OnDateSetListener to create a standard date picker with a listener:

```
public class DatePickerFragment extends DialogFragment
        implements DatePickerDialog.OnDateSetListener {
}
```

It adds automatically the following in the import block at the top:

```
import android.app.DatePickerDialog.OnDateSetListener;
import android.support.v4.app.DialogFragment;
```

- OR
- Android Studio also shows a red light bulb icon in the left margin, prompting you to implement methods. Click the icon and, with **onDateSet** already selected and the "Insert @Override" option checked, click **OK** to create the empty onDateSet() method.
- Android Studio then automatically adds the following in the import block at the top:

```
import android.widget.DatePicker;
```

- Replace onCreateView() with onCreateDialog():

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
}
```

- When you extend DialogFragment, you should override the onCreateDialog() callback method rather than onCreateView. You use your version of the callback method to initialize the year, month, and day for the date picker.

Setting the defaults and returning the picker

- To set the default date in the picker and return it as an object you can use, follow these steps:
- Add the following code to the onCreateDialog() method to set the default date for the picker:

```
// Use the current date as the default date in the picker.
final Calendar c = Calendar.getInstance();
int year = c.get(Calendar.YEAR);
int month = c.get(Calendar.MONTH);
int day = c.get(Calendar.DAY_OF_MONTH);
```

- As you enter Calendar, you are given a choice of which Calendar library to import. Choose this one:

```
import java.util.Calendar;
```

- The Calendar class sets the default date as the current date—it converts between a specific instant in time and a set of calendar fields such as YEAR, MONTH, DAY_OF_MONTH, HOUR, and so on.
- Calendar is locale-sensitive, and its class method getInstance() returns a Calendar object whose calendar fields have been initialized with the current date and time.
- Add the following statement to the end of the method to create a new instance of the date picker and return it:

```
return new DatePickerDialog(getActivity(), this, year, month, day);
```

Showing the picker

- In the Main Activity, you need to create a method to show the date picker. Follow these steps:
- Create a method to instantiate the date picker dialog fragment:

```
public void showDatePickerDialog(View v) {
    DialogFragment newFragment = new DatePickerFragment();
    newFragment.show(getSupportFragmentManager(), "datePicker");
}
```

- You can then use showDatePickerDialog() with the android:onClick attribute for a button or other input control:

```
<Button
    android:id="@+id/button_date"
    ...
    android:onClick="showDatePickerDialog" />
```

Processing the user's picker choice

- The onDateSet() method is automatically called when the user makes a selection in the date picker, so you can use this method to do something with the chosen date. Follow these steps:
- To make the code more readable, change the onDateSet() method's parameters from int i, int i1, and int i2 to int year, int month, and int day:

```
public void onDateSet(DatePicker view, int year, int month, int day) {
```

Open MainActivity and add

the processDatePickerResult() method signature that takes the year, month, and day as arguments:

```
public void processDatePickerResult(int year, int month, int day)
{
}
```

- Add the following code to the processDatePickerResult() method to convert the month, day, and year to separate strings:

```
String month_string =Integer.toString(month+1);
String day_string = Integer.toString(day);
String year_string = Integer.toString(year);
```

- **Here** The month integer returned by the date picker starts counting at 0 for January, so you need to add 1 to it to start show months starting at 1.
- Add the following after the above code to concatenate the three strings

```
String dateMessage = (month_string + "/" + day_string + "/" + year_string);
```

- Add the following after the above statement to display a Toast message:

```
Toast.makeText(this, "Date: " + dateMessage,
                Toast.LENGTH_SHORT).show();
```

- Extract the hard-coded string "Date: " into a string resource named date. This automatically replaces the hard-coded string with getString(R.string.date). The code for the processDatePickerResult() method should now look like this:

```
public void processDatePickerResult(int year, int month, int day) {
  String month_string = Integer.toString(month + 1);
  String day_string = Integer.toString(day);
  String year_string = Integer.toString(year);
  // Assign the concatenated strings to dateMessage.
  String dateMessage = (month_string + "/" + day_string + "/" + year_string);
Toast.makeText(this, getString(R.string.date) + dateMessage,
                Toast.LENGTH_SHORT).show();

}
```

- Now open **DatePickerFragment**, and add the following to the onDateSet() method to invoke the processDatePickerResult() method in MainActivity and pass it the year, month, and day:

```
public void onDateSet(DatePicker view, int year, int month, int day) {
  // Set the activity to the Main Activity.
  MainActivity activity = (MainActivity) getActivity();
  // Invoke Main Activity's processDatePickerResult() method.
  activity.processDatePickerResult(year, month, day);
}
```

- You use getActivity() which, when used in a fragment, returns the activity the fragment is currently associated with. You need this because you can't call a method in MainActivity without the context of MainActivity (you would have to use an intent instead, as you learned in a previ

lesson). The activity inherits the context, so you can use it as the context for calling the method (as in activity.processDatePickerResult).

Using the same procedures for the time picker

- Add a blank fragment called TimePickerFragment that extends DialogFragment and implements TimePickerDialog.OnTimeSetListener:

```
public class TimePickerFragment extends DialogFragment
        implements TimePickerDialog.OnTimeSetListener {
```

- Add with @Override a blank onTimeSet() method:
- Android Studio also shows a red light bulb icon in the left margin, prompting you to implement methods.
- Click the icon and, with **onTimeSet** already selected and the "Insert @Override" option checked, click **OK** to create the empty onTimeSet() method. Android Studio then automatically adds the following in the import block at the top:
- import android.widget.TimePicker;
- Use onCreateDialog() to initialize the time and return the dialog:

```
public Dialog onCreateDialog(Bundle savedInstanceState) {
  // Use the current time as the default values for the picker.
  final Calendar c = Calendar.getInstance();
  int hour = c.get(Calendar.HOUR_OF_DAY);
  int minute = c.get(Calendar.MINUTE);

  // Create a new instance of TimePickerDialog and return it.
  return new TimePickerDialog(getActivity(), this, hour, minute,
      DateFormat.is24HourFormat(getActivity()));
}
```

- Now Show the picker: Open **MainActivity** and create a method to instantiate the date picker dialog fragment:

```
public void showDatePickerDialog(View v) {
    DialogFragment newFragment = new DatePickerFragment();
    newFragment.show(getSupportFragmentManager(), "datePicker");
}
```

- Now Use showDatePickerDialog() with the android:onClick attribute for a button or other input control:

```
<Button
    android:id="@+id/button_date"
    ...
    android:onClick="showDatePickerDialog"/>
```

- Create the processTimePickerResult() method in **MainActivity** to process the result of choosing from the time picker:

```
public void processTimePickerResult(int hourOfDay, int minute) {
```

```
// Convert time elements into strings.
String hour_string = Integer.toString(hourOfDay);
String minute_string = Integer.toString(minute);
// Assign the concatenated strings to timeMessage.
String timeMessage = (hour_string + ":" + minute_string);
Toast.makeText(this, getString(R.string.time) + timeMessage,
                    Toast.LENGTH_SHORT).show();
}
```

- Now Use onTimeSet() to get the time and pass it to the processTimePickerResult() method in MainActivity:

```
public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
  // Set the activity to the Main Activity.
  MainActivity activity = (MainActivity) getActivity();
  // Invoke Main Activity's processTimePickerResult() method.
  activity.processTimePickerResult(hourOfDay, minute);
}
```

Syllabus Topic : Menus

2.6 Menus

- Menus are a common user interface component in many types of applications. To provide a familiar and consistent user experience, you should use the Menu APIs to present user actions and other options in your activities.
- We are creating a simple menu with 6 menu items. On clicking on single menu item a simple Toast message will be shown.
  1. Create a new project **File ⇒ New ⇒ Android Project** and give activity name as **AndroidMenusActivity**.
  2. Now create an XML file under res/layout folder and name it as menu.xml.
  3. Open menu.xml file and type following code. In the following code we are creating a single menu with 6 menu items. Each menu item has an icon and title for display the label under menu icon. Also we have id for each menu item to identify uniquely.

☞ **menu.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <!-- Single menu item
    Set id, icon and Title for each menu item   -->
  <item android:id="@+id/menu_bookmark"
    android:icon="@drawable/icon_bookmark"
    android:title="Bookmark" />
```

```
<item android:id="@+id/menu_save"
  android:icon="@drawable/icon_save"
  android:title="Save" />
<item android:id="@+id/menu_search"
  android:icon="@drawable/icon_search"
  android:title="Search" />
<item android:id="@+id/menu_share"
  android:icon="@drawable/icon_share"
  android:title="Share" />
<item android:id="@+id/menu_delete"
  android:icon="@drawable/icon_delete"
  android:title="Delete" />
<item android:id="@+id/menu_preferences"
  android:icon="@drawable/icon_preferences"
  android:title="Preferences" />
</menu>
```

- Now open your main Activity class file (AndroidMenusActivity.java) and type following code. In the following code each menu item is identified by its ID in switch case statement.
- Finally run your project by **right clicking on your project folder ⇒ Run As ⇒ 1 Android Application** to test your application. Android Emulator click on Menu Button to launch menu.

Android1.\1'nusActiity.jaa

```
packagecom.androidhive.androidmenus;
import android.app.Activity;
import android.os.Bundle;
 import android.view.Menu;
import android.view.MenuInflater;
import android.view.Menultem;
import android.widget.Toast;
public class AndroidMenusActivity extends Activity {
@override
public void onCreate{Bundle savedinstanceState )
   {super.onCreate{savedinstanceState);setContentView{R.layout.main);
}

//InitiatingMenuXMLfile{menu.xml)
@override
Public Boolean onCreateOptionsMenu{Menu menu)
{
MenuInflater menuInflater=getMenuinflater{);menuInflater.inflate{R.layout.menu,menu);
return true;
}
/**
```

```
*Event Handling for Individual menu item selected
*Identify single menu item by it1sid
**/
@override
Public Boolean onOptionsItemSelected(MenuItem item)
{
switch(item.getitemid())
{
case R.id.menubook-mark:
//Single menu item is selected do something
//Ex: launching new activity/screen or shewalert meassage
Toast.makeText(AndroidMenusActivity.this,"Book-mark is Selected",
Toast.LENGTH_SHORT).show();
return true;
case R.id.menusave:
Toast.makeText(AndroidMenusActivity.this, "Save is Selected",Toast.LENGTH_SHORT).show();
return true;

case R.id.menusearch:
Toast.makeText(AndroidMenusActivity.this,"Search is Selected",
Toast.LENGTH_SHORT).show();
```

Types of Menu in android

- A menu is a set of options the user can select from to perform a function, such as searching for information, saving information, editing information, or navigating to a screen.
- Android offers the following types of menus, which are useful for different situations (Refer to the Fig. 2.6.1).

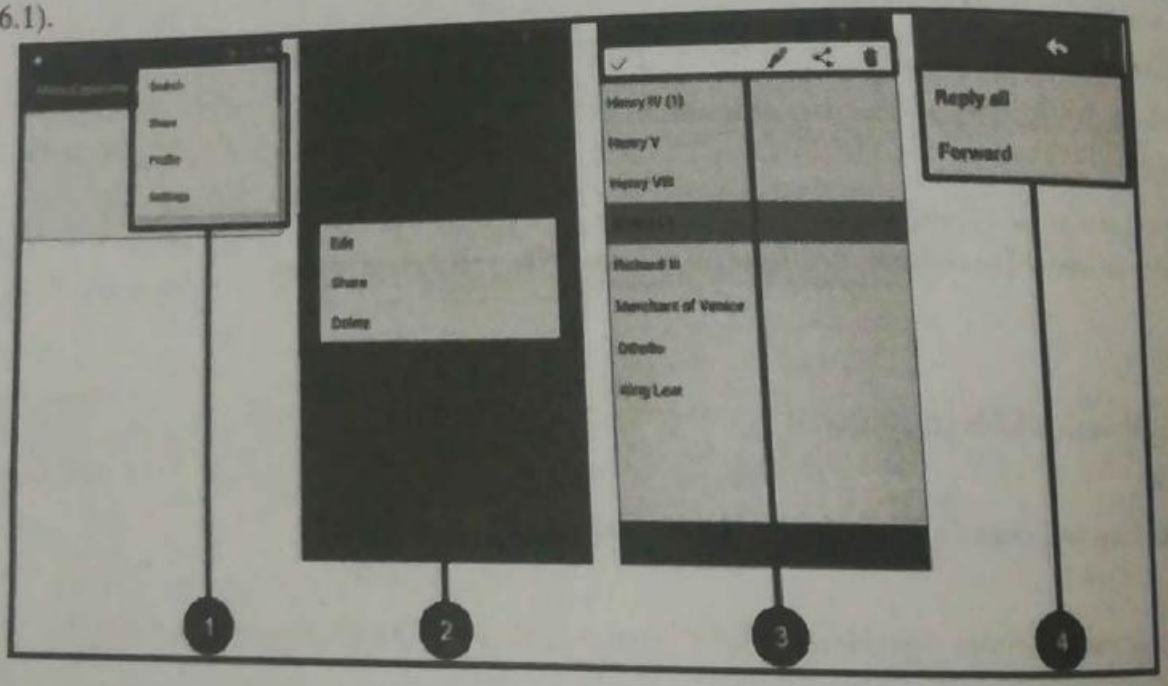


Fig. 2.6.1

1. Options menu

- Appears in the app bar and provides the primary options that affect using the app itself.
- Examples of menu options: **Search** to perform a search, **Bookmark** to save a link to a screen, and **Settings** to navigate to the Settings screen.

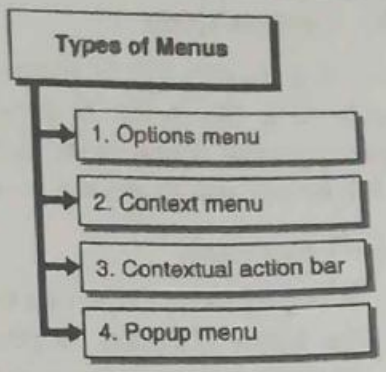


Fig. C2.2 : Types of Menus

2. Context menu

Appears as a floating list of choices when the user performs a long tap on an element on the screen. Examples of menu options: **Edit** to edit the element, **Delete** to delete it, and **Share** to share it over social media.

3. Contextual action bar

Appears at the top of the screen overlaying the app bar, with action items that affect the selected element(s). Examples of menu options: **Edit**, **Share**, and **Delete** for one or more selected elements.

4. Popup menu

- Appears anchored to a view such as an ImageButton, and provides an overflow of actions or the second part of a two-part command. Example of a popup menu: the Gmail app anchors a popup menu to the app bar for the message view with **Reply**, **Reply All**, and **Forward**.
- Android offers an easy programming interface for developers to provide standardized application menus for various situations.

Android offers three fundamental types of application menus

(A) Options Menu

- This is the primary set of menu items for an Activity. It is revealed by pressing the device MENU key.
- It's where you should place actions that have a global impact on the app, such as "Search," "Compose email," and "Settings."Within the Options Menu are two groups of menu items.

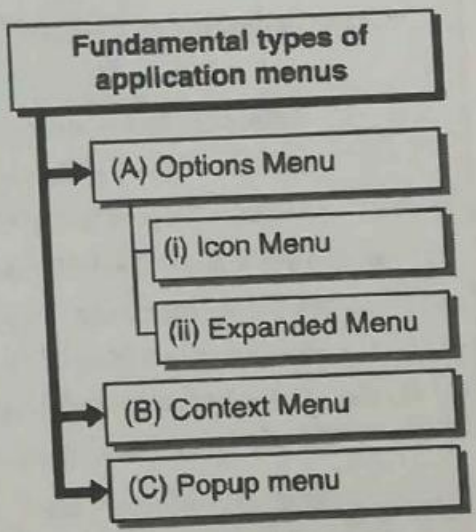


Fig. C2.3: Types of apps menus

(i) Icon Menu

This is the collection of items initially visible at the bottom of the screen at the press of the MENU key. It supports a maximum of six menu items. These are the only menu items that support icons and the only menu items that do not support checkboxes or radio buttons.

(ii) Expanded Menu

This is a vertical list of items exposed by the "More" menu item from the Icon Menu. It exists only when the Icon Menu becomes over-loaded and is comprised of the sixth Option Menu item and the rest.

➜ **(B) Context Menu**

This is a floating list of menu items that may appear when you perform a long-press on a View (such as a list item). It provides actions that affect the selected content or context frame. The contextual action mode displays action items that affect the selected content in a bar at the top of the screen and allows the user to select multiple items.

➜ **(C) Popup menu**

- A popup menu displays a list of items in a vertical list that's anchored to the view that invoked the menu. It's good for providing an overflow of actions that relate to specific content or to provide options for a second part of a command.
- Actions in a popup menu should **not** directly affect the corresponding content—that's what contextual actions are for. Rather, the popup menu is for extended actions that relate to regions of content in your activity.

2.6.1 Menu Icons

- Final icon must be exported as a transparent PNG file. Do not include a background color.
- The size of the icon should be like this.

high-density (hdpi)	medium-density (mdpi)	low-density (ldpi)
Full Asset: 72 x 72 px Icon: 48 x 48 px Square Icon: 44 x 44 px	Full Asset: 48 x 48 px Icon: 32 x 32 px Square Icon: 30 x 30 px	Full Asset: 36 x 36 px, Icon: 24 x 24 px, Square Icon: 22 x 22 px Action Bar

- If we modify one of the lines of the menu file **res/menu/my_menu.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">

  <item android:id="@+id/stop"
        android:title="@string/stop"
        android:orderInCategory="3"
        android:icon="@drawable/eject"
        android:showAsAction="ifRoom|withText" />
  <item android:id="@+id/dumb"
        android:orderInCategory="2"
        android:title="dumb" />
  <item android:id="@+id/disabled"
        android:orderInCategory="4"
        android:enabled="false"
        android:title="@string/disabled"/>
 <group android:id="@+id/group_items"
   android:menuCategory="secondary"
       android:visible="false">
   <item android:id="@+id/group_item1"
      android:title="@string/group_item1" />
```

```
    <item android:id="@+id/group_item2"
        android:title="@string/group_item2" />
</group>
<item android:id="@+id/submenu"
    android:orderInCategory="3"
    android:title="@string/submenu">
    <menu>
        <item android:id="@+id/do_something"
            android:title="@string/do_something"
            android:visible="true"
            android:alphabeticShortcut="s" />
        <item android:id="@+id/do_nothing"
            android:title="@string/do_nothing"
            android:visible="false"
            android:alphabeticShortcut="n" />
    </menu>
</item>
</menu>
```

- We can see the menu icon on the right corner of the screen, and it should look like this.

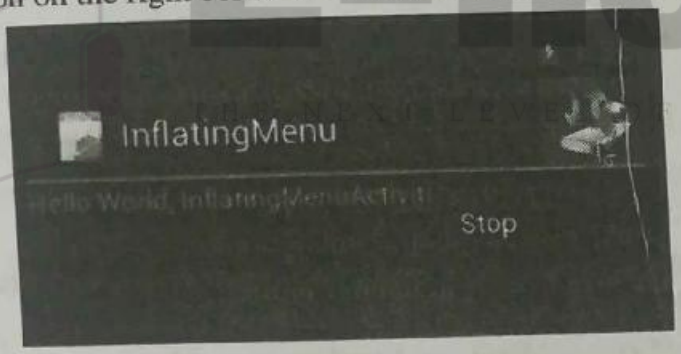


Fig. 2.6.2

2.6.2(A) Options Menu

- The **Options Menu** is an Android user interface component that provides standardized menus. It is opened by pressing the device **MENU** key.
- The Options Menu include basic application functions and any necessary navigation items (e.g., to a home screen or application settings).Add Submenus for organizing topics and including extra menu functionality.
- When this menu is opened for the first time, the Android system will call the Activity onCreateOptionsMenu() callback method.
- Override this method in our Activity and populate the Menu object given to us. and populate the menu by inflating a menu resource that was defined in XML, or by calling add() for each item we'd like in the menu. This method adds a MenuItem, and returns the newly created object to you.
- We can use the returned MenuItem to set additional properties like an icon, a keyboard shortcut, an intent, and other settings for the item.

- There are multiple add() methods. use one that accepts an itemId argument. This is a unique integer that allows us to identify the item during a callback.
- When a menu item is selected from the Options Menu, we'll receive a callback to the onOptionsItemSelected() method of our Activity. This callback passes us the MenuItem that has been selected.
- To identify the item by requesting the itemId, with getItemId(), which returns the integer that was assigned with the add() method. Once we identify the menu item, we can take the appropriate action.

☞ **Creating an Options Menu**

- Rather than building activity's options menu during onCreate(), the way we wire up the rest of our UI, we instead need to implement onCreateOptionsMenu(). This callback receives an instance of Menu.
- The first thing we should do is chain upward to the superclass (super.onCreateOptionsMenu(menu)), so the Android framework can add in any menu choices it feels are necessary. Then we can go about adding our own options.
- If we will need to adjust the menu during our activity's use such as disable a now-invalid menu choice, just hold onto the Menu instance we receive in onCreateOptionsMenu().
- Alternatively, we can implement onPrepareOptionsMenu(), which is called just before displaying the menu each time it is requested.

☞ **Adding Menu**

- Given that we have received a Menu object via onCreateOptionsMenu(), we add menu choices by calling add().
- Following method, which require some combination of the parameters as following:
- **1. A group identifier (groupId):** This should be NONE unless we are creating a specific grouped set of menu choices for use with setGroupCheckable().
- **2. A choice identifier (itemId):** This is for use in identifying this choice in the onOptionsItemSelected() callback when a menu choice is selected.
- Here is an example for Options Menu and handling item selections:

```
package com.bogotobogo.OptionsMenu;
import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
public class OptionsMenuActivity extends Activity {
   private static final int MENU_NEW_GAME = Menu.FIRST;
   private static final int MENU_QUIT = Menu.FIRST + 1;
    /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.main);
  }
```

```
/* Creates the menu items */
public boolean onCreateOptionsMenu(Menu menu) {
menu.add(0, MENU_NEW_GAME, 0, "New Game");
menu.add(0, MENU_QUIT, 0, "Quit");
return true;
}
/* Handles item selections */
public boolean onOptionsItemSelected(MenuItem item) {
switch (item.getItemId()) {
    case MENU_NEW_GAME:
        newGame();
        return true;
    case MENU_QUIT:
        quit();
        return true;
}
return false;
}
public void newGame() {}
public void quit() {}
}
```


- The add() method used in this sample takes four arguments: groupId, itemId, order, and title.
- The groupId allows you to associate this menu item with a group of other items, in this example though, we ignore it. itemId is a unique integer that we give the MenuItem so that can identify it in the next callback. order allows us to define the display order of the item, by default, they are displayed by the order in which we add them. title is, of course, the name that goes on the menu item (this can also be a string resource, and we recommend you do it that way for easier localization).
- Here is the result of the run (Fig. 2.6.3).

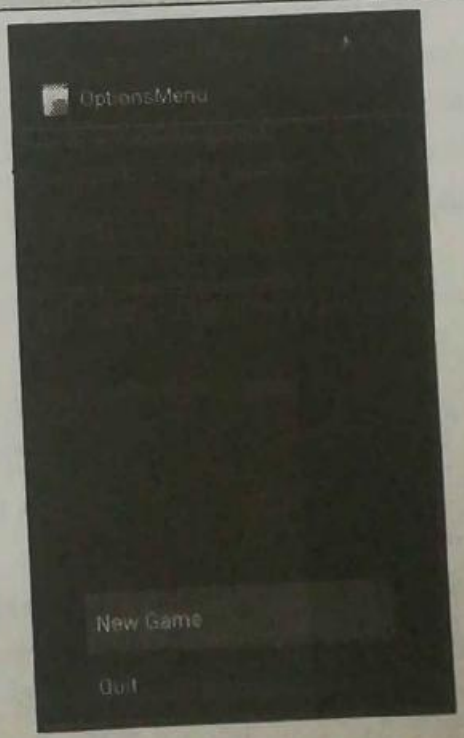


Fig. 2.6.3

Adding Icons

Icons can also be added to items that appears in the Icon Menu with setIcon(). For example, if we modify one of the line in the Java code above after put icon into res/drawable/:

```
menu.add(0, MENU_QUIT, 0, "Quit") ->
menu.add(0, MENU_QUIT, 0, "Quit").setIcon(R.drawable.ic_quit);
```

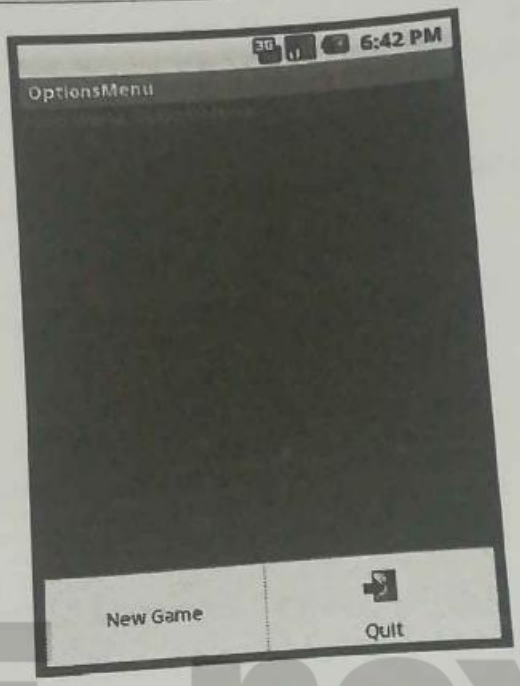


Fig. 2.6.4

```
package com.bogotobogo.OptionsMenu;
import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;

public class OptionsMenuActivity extends Activity {

  private static final int MENU_NEW_GAME = Menu.FIRST;
  private static final int MENU_QUIT = Menu.FIRST + 1;
    /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
  }
```

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
MenuInflater inflater = getMenuInflater();
inflater.inflate(R.menu.my_menu, menu);
return true;
}
/* Handles item selections */
public boolean onOptionsItemSelected(MenuItem item) {
switch (item.getItemId()) {
    case MENU_NEW_GAME:
        newGame();
        return true;
    case MENU_QUIT:
        quit();
        return true;
}
return false;
}
public void newGame() {}
public void quit() {}
}
```

with my_menu.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/new_game"
    android:title="@string/new_game" />
  <item android:id="@+id/quit"
    android:icon="@drawable/ic_quit"
    android:title="@string/quit" />
</menu>
```

and strings.xml

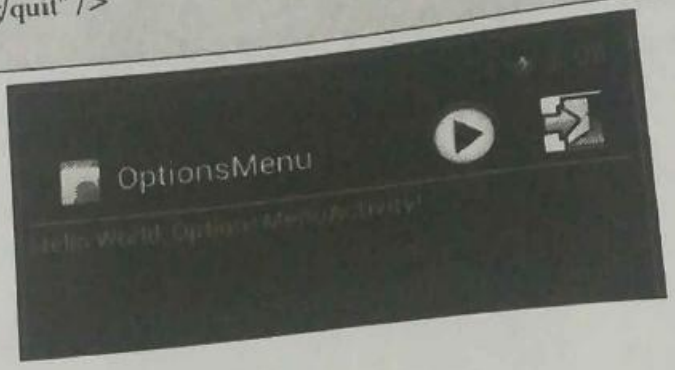
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="hello">Hello World, OptionsMenuActivity!</string>
  <string name="app_name">OptionsMenu</string>
  <string name="new_game">New Game</string>
  <string name="quit">Quit</string>
</resources>
```

if we use Action Bar, we can display the menu icons:

And with modified my_menu.xml:

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

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/new_game"
     android:icon="@drawable/ic_new"
     android:showAsAction="ifRoom|withText"
     android:title="@string/new_game" />
  <item android:id="@+id/quit"
     android:icon="@drawable/ic_quit"
     android:showAsAction="ifRoom|withText"
     android:title="@string/quit" />
</menu>
```



➜ 2.6.2(B) Context Menu

- Context Menu revealed with a "right-click" on a PC. When a view is registered to a context menu, performing a "long-press on the object will reveal a floating menu that provides functions relating to that item.
- Context menus can be registered to any View object, however, they are most often used for items in a ListView, which helpfully indicates the presence of the context menu by transforming the background color of the ListView item when pressed.
- Here is our Java code for the ContextMenu example.

```
package com.bogotobogo.ContextMenuA;
import android.os.Bundle;
import android.app.Activity;
import android.view.ContextMenu;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.ContextMenu.ContextMenuInfo;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;
public class ContextMenuAActivity extends Activity {
   String[] phones={
         "HTC Rezound", "Samsung Galaxy S II Skyrocket",
```

```
        "Samsung Galaxy Nexus", "Motorola Droid Razr",
        "Samsung Galaxy S", "Samsung Epic Touch 4G",
        "¡Phone 4S", "HTC Titan"
        };
private static final int MENU_NEW_GAME = Menu.FIRST;
private static final int MENU_QUIT = Menu.FIRST + 1;
private static final int MENU_EDIT = Menu.FIRST + 2;
private static final int MENU_DELETE = Menu.FIRST + 3;

@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
ListView list = (ListView)findViewById(R.id.list);
ArrayAdapter<String> adapter =
            new ArrayAdapter<String>(this, R.layout.listitem, phones);
        list.setAdapter(adapter);
list.setAdapter(adapter);
registerForContextMenu(list);
}
 @Override
 public void onCreateContextMenu(ContextMenu menu, View v,
                                 ContextMenuInfo menuInfo) {
        if (v.getId()==R.id.list) {
            AdapterView.AdapterContextMenuInfo info
                = (AdapterView.AdapterContextMenuInfo)menuInfo;
            menu.setHeaderTitle(phones[info.position]);
            menu.add(0, MENU_EDIT, 0, "Edit");
            menu.add(0, MENU_DELETE, 0,  "Delete");
        }
 }
/* Creates the menu items */
@Override
public boolean onCreateOptionsMenu(Menu menu) {
 menu.add(0, MENU_NEW_GAME, 0, "New Game");
 menu.add(0, MENU_QUIT, 0, "Quit").setIcon(R.drawable.ic_quit);
 return true;
}
/* Handles item selections */
@Override
public boolean onOptionsItemSelected(MenuItem item) {
```

```
switch (item.getItemId()) {
        case MENU_NEW_GAME:
            return true;
        case MENU_QUIT:
            return true;
}
return false;
}
@Override
public boolean onContextItemSelected(MenuItem item) {
 TextView text = (TextView)findViewById(R.id.footer);
 switch (item.getItemId()) {
        case MENU_EDIT:
            text.setText("Edit selected");
            return true;
        case MENU_DELETE:
            text.setText("Delete selected");
            return true;
        default:
            return super.onContextItemSelected(item);
    }
  }
}
```

- To create a context menu, you must override the Activity's context menu callback methods: **onCreateContextMenu**() and **onContextItemSelected**().
- Inside the onCreateContextMenu() callback method, you can add menu items using one of the **add**() methods, or by inflating a menu resource that was defined in XML. Then, register a ContextMenu for the View, with **registerForContextMenu**().
- In **onCreateContextMenu**(), we are given not only the ContextMenu to which we will add menu but also the **View** that was selected and a **ContextMenuInfo** object, which provides additional information about the object that was selected. In this example, nothing special is done in **onCreateContextMenu**(), just a couple items are added as usual.
- In the **onContextItemSelected**() callback, we request the **getItemId**() from the MenuItem, which provides information about the currently selected item.
- All we need from this is the list ID for the selected item, so whether editing a note or deleting it. This ID can passed to the "edit()" or "delete" methods though they are not implemented in this example. Here, we are just writing a text to the footer to tell the user which menu has been selected.
- Note that we register this context menu for all the items in a ListView. Then, we pass the entire **ListView** to the **registerForContextMenu(View)** method:

```
ListView list = (ListView)findViewById(R.id.list);
```

```
ArrayAdapter<String> adapter =
        new ArrayAdapter<String>(this, R.layout.listitem, phones);
list.setAdapter(adapter);
registerForContextMenu(list);
```

With layout file: main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="vertical">
<ListView
android:id="@+id/list"
android:layout_width="fill_parent"
android:layout_height="0px"
android:layout_weight="1"/>
<TextView
android:id="@+id/footer"
android:layout_width="fill_parent"
android:layout_height="60dip"
android:text="@string/footer"
android:padding="4dip"
android:background="#FF666666"/>
</LinearLayout>
```

and listitem.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<TextView
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:textSize="24dip"
android:padding="8dip"/>
```

- The "footer" string used in main.xml is defined in res/values/strings.xml.

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

```
<string name="hello">Hello World, ContextMenuA!</string>
<string name="app_name">ContextMenuA</string>
<string name="footer">Long click to get context menu</string>
</resources>
```

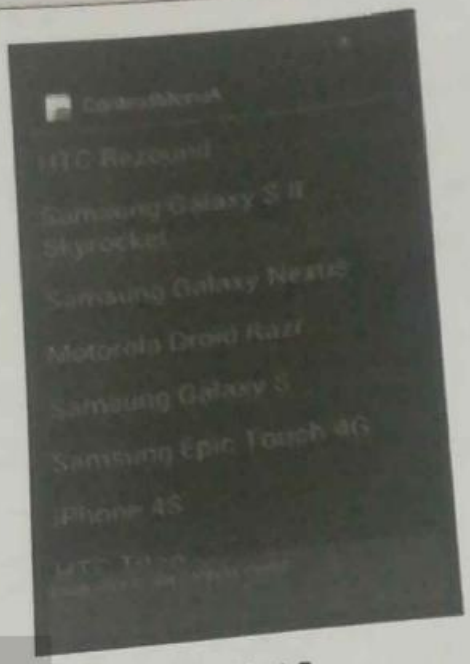
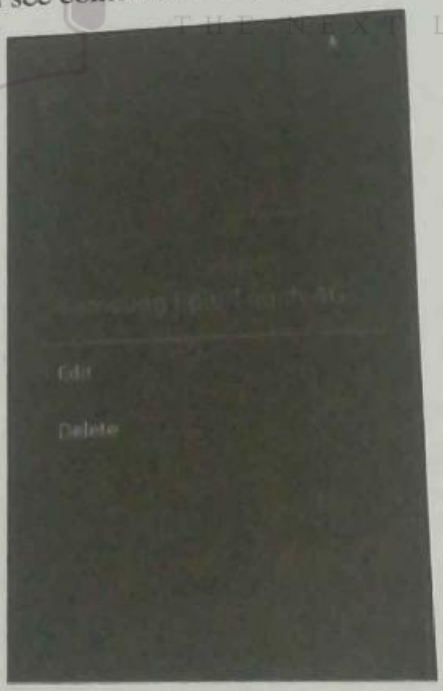
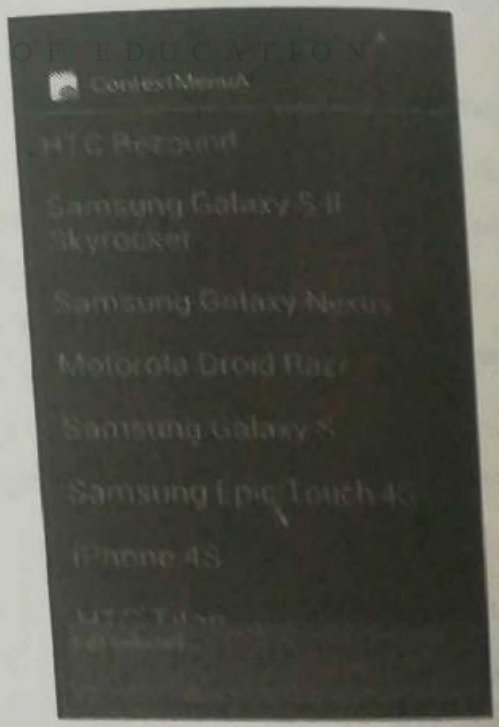


Fig. 2.6.5

- Notice the lower part of screen which is "footer" is used as a space for a message. At the long click we should see context menu.(Fig. 2.6.6(a)).





(a) **(b)**

Fig. 2.6.6

- We do not have any specific actions at the selection on the context menu, but it shows what has been made in the footer section. In this case, the message "Edit selected".(Fig. 2.6.6(b))

- Develop an application for working with Menus and Screen Navigation.

XML Code

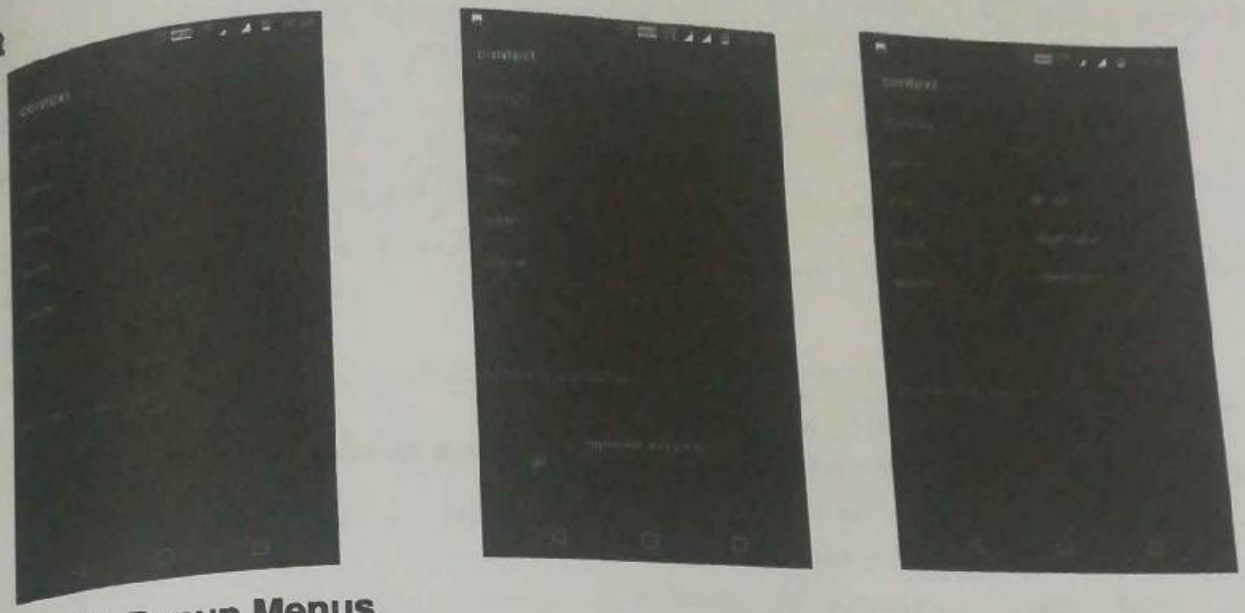
```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.kbp.context.MainActivity">

    <LinearLayout
        android:layout_width="368dp"
        android:layout_height="495dp"
        android:orientation="vertical"
        tools:layout_editor_absoluteY="8dp"
        tools:layout_editor_absoluteX="8dp">
        <ListView
            android:id="@+id/LV1"
            android:layout_width="match_parent"
            android:layout_height="343dp" />
        <TextView
            android:id="@+id/tv1"
            android:layout_width="363dp"
            android:layout_height="101dp"
            android:text="LONG PRESS IN ANY ITEM" />
    </LinearLayout>
</android.support.constraint.ConstraintLayout>
```

Java Code

```
package com.example.kbp.context;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.ContextMenu;
import android.view.MenuItem;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity
{
```

```
ListView listView;
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    listView = (ListView) findViewById(R.id.LV1);
    String[] planets = {"mercury", "venus", "mars", "earth", "jupiter"};
    ArrayAdapter adapter = new ArrayAdapter(this, android.R.layout.simple_list_item_1, planets);
    listView.setAdapter(adapter);
    registerForContextMenu(listView);
}
@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo
menuInfo)
{
    super.onCreateContextMenu(menu, v, menuInfo);
    menu.setHeaderTitle("select the action");
    menu.add(0,v.getId(),0,"delete");
    menu.add(0,v.getId(),0,"uppercase");
    menu.add(0,v.getId(),0,"lowercase");
}
@Override
public boolean onContextItemSelected(MenuItem item)
{
    if(item.getTitle()=="delete")
    {
        Toast.makeText(this,"Delete was press", Toast.LENGTH_SHORT).show();
    }
    else if(item.getTitle()=="uppercase")
    {
        Toast.makeText(this,"uppercase was press", Toast.LENGTH_SHORT).show();
    }
    else if(item.getTitle()=="lowercase")
    {
        Toast.makeText(this,"lowercase was press", Toast.LENGTH_SHORT).show();
    }
    return true;
}
}
```

Output



2.6.3(C) Popup Menus

- Popup menus are useful for displaying extended options associated with a specific action.
- For eg: 'send' is action and have multiple extended options, like 'send by email' or 'send by sms' etc.
- Context menus they can be invoked by any event such a button click not just long clicks. They are associated with the specific view that invoked it and each view in an activity can have its own popup window.

Steps to create a Popup menu

- Create a new instance of PopupMenu class and pass the instance of the Context (usually the current activity) and the view (for which the pop-up menu is desired) as arguments.
- Inflate the menu resource using:
- popupMenu.inflate () if you are using Android 4.0 SDK and above (or)
- MenuInflater class (popupMenu.getMenuInflater().inflate() method) if you are using Android 3.0 SDK
- Call popupMenu.show() to display the menu

Responding to menu item selections

- Override the popupMenu.setOnMenuItemClickListener () method and provide a call back for handling user selections. The use selected menu item is passed in as argument to the callback method.

Example

- Single button that is wrapped inside a linear layout. Clicking the button invokes a popup menu that lets us change the background color for the button as shown in the sample screenshot.

Menu resource File

- This is the same color menu resource file under res/menu folder that we used in android context menu example.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/menu_red" android:title="Red" />
  <item android:id="@+id/menu_green"
android:title="Green"/>
  <item android:id="@+id/menu_blue" android:title="Blue"/>
</menu>
```

☞ **Layout definition file**

– One single button within a linear layout.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
<Button android:id="@+id/popupMenuBtn"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="Show me the Popup"
        android:layout_gravity="center"
        android:gravity="center"/>
</LinearLayout>
```

☞ **Java Source Code**

– The button element acts as the view responsible for invoking the popup menu. Create a new instance of the popup menu and pass the activity and the button as arguments

– Inflate the menu resource and associate it with the popup menu instance.

– Call popupmenu.show () method within the onClick listener to display the menu when the user clicks the button.

– Override the popupMenu.setOnMenuItemClickListener () method to change the background color of the button based on user selections.


```
public class MenuDemo extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.menudemo);
```

```
/**
 * A Button that acts as the view element for the popup menu.
 */
final Button btn = (Button) findViewById(R.id.popupMenuBtn);
/**
 * Step 1: Create a new instance of popup menu
 */
final PopupMenu popupMenu = new PopupMenu(this, btn);
/**
 * Step 2: Inflate the menu resource. Here the menu resource is
 * defined in the res/menu project folder
 */
popupMenu.inflate(R.menu.color_menu);
/**
 * Step 3: Call show() method on the popup menu to display the
 * menu when the button is clicked.
 */
btn.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View v) {
    popupMenu.show();
  }
});
/**
 * Handle menu item clicks
 */
popupMenu.setOnMenuItemClickListener(
    new PopupMenu.OnMenuItemClickListener() {
  @Override
  public boolean onMenuItemClick(MenuItem item) {
    switch (item.getItemId()) {
      case R.id.menu_red:
        btn.setBackgroundResource(R.color.LightRed);
        break;
      case R.id.menu_blue:
        btn.setBackgroundResource(R.color.DullBlue);
        break;
      case R.id.menu_green:
        btn.setBackgroundResource(R.color.LightGreen);
        break;
    }
```

```
        return true;
      }
    });
  } }
```

☞ **xml code**

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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"
  tools:context="com.example.kbp.menu.MainActivity">
  <Button
    android:id="@+id/btn1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentEnd="true"
    android:layout_alignParentLeft="true"
    android:layout_alignParentRight="true"
    android:layout_alignParentStart="true"
    android:text="Button" />
  <TextView
    android:id="@+id/tv1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_alignParentEnd="true"
    android:layout_alignParentLeft="true"
    android:layout_alignParentRight="true"
    android:layout_alignParentStart="true"
    android:layout_marginBottom="100dp"
    android:layout_marginLeft="10dp"
    android:layout_marginRight="10dp"
    android:text="Click The Button"
    android:textSize="35sp" />
</RelativeLayout>
```

☞ **Java Code**

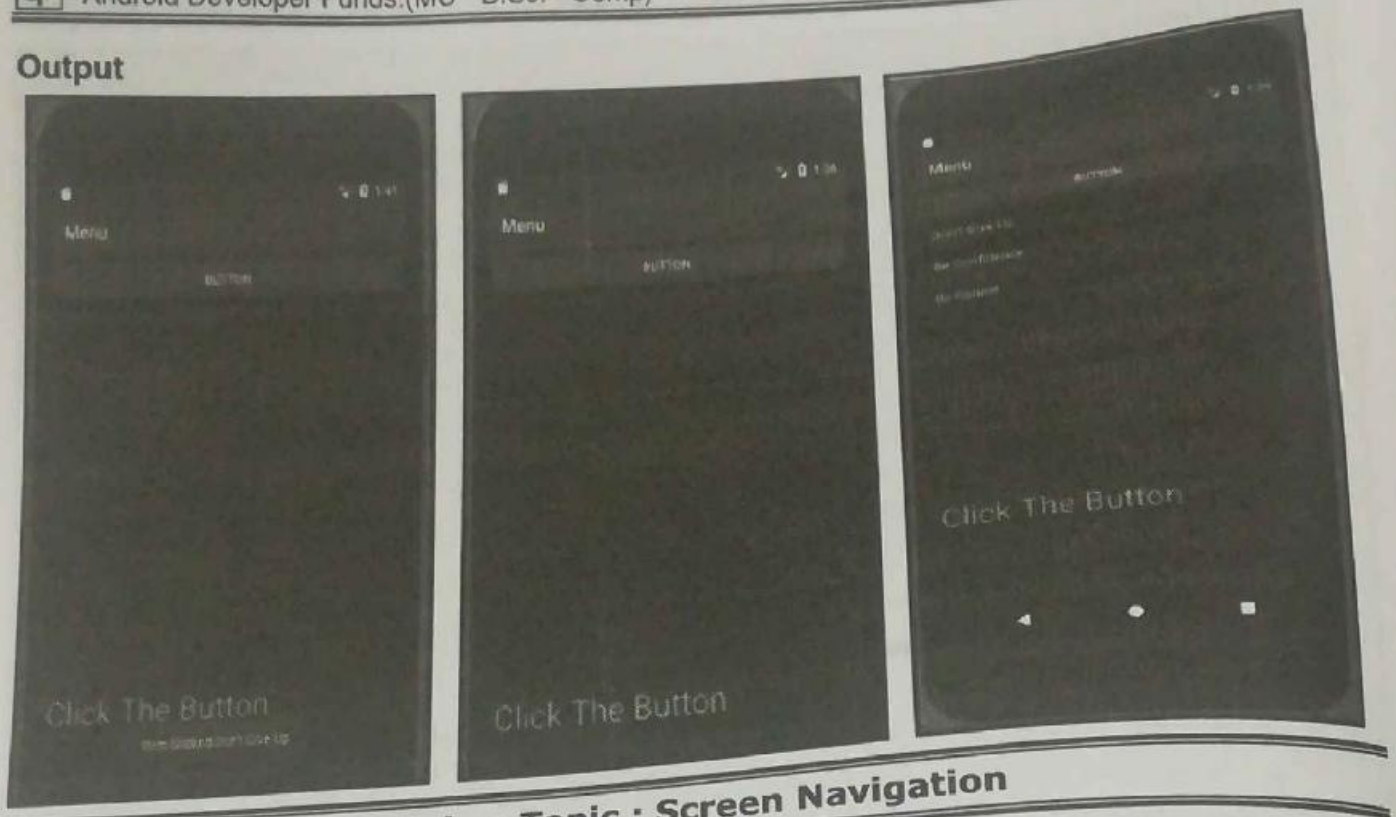
```
package com.example.kbp.menu;
import android.support.v7.app.AppC...
```

```
import android.support.v7.widget.PopupMenu;
import android.os.Bundle;
import android.widget.Button;
import android.widget.Toast;
import android.view.View;
import android.view.MenuItem;
import android.view.Menu;
public class MainActivity extends AppCompatActivity {
  Button btnpopmenu;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    btnpopmenu=(Button)findViewById(R.id.btn1);
    btnpopmenu.setOnClickListener(new View.OnClickListener(){

      @Override
      public void onClick(View v) {
        PopupMenu popupMenu=new PopupMenu(MainActivity.this,btnpopmenu);
        popupMenu.getMenuInflater().inflate(R.menu.menu_main,popupMenu.getMenu());
        popupMenu.setOnMenuItemClickListener(new PopupMenu.OnMenuItemClickListener() {
          @Override
          public boolean onMenuItemClick(MenuItem item) {
            return true;
          }
        });
        popupMenu.show();
      }
    });
  } }
```

Menu_Main Created Xml File

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/Sandesh" android:title="Vidhate"/>
  <item android:id="@+id/Ajinkya" android:title="BHise"/>
  <item android:id="@+id/Mane_Madam" android:title="Aswini_Madam"/>
</menu>
```

Output



Syllabus Topic : Screen Navigation

2.7 Screen Navigation

- Determine the paths users should take through your app in order to do something, such as placing an order or browsing through content.
- Each path enables users to navigate across, into, and back out from the different tasks and pieces of content within the app.
- In many cases need several different paths through your app that offer the following types of navigation.

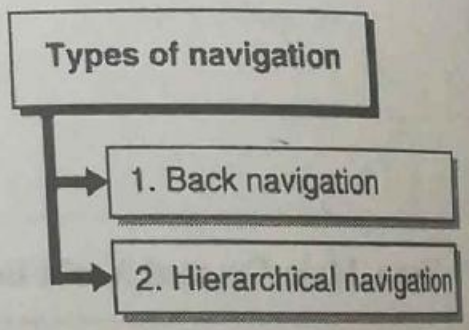


Fig. C.2.4 : Types of navigation

➜ **1. Back navigation**

Users can navigate back to the previous screen using the Back button.

➜ **2. Hierarchical navigation**

Users can navigate through a hierarchy of screens organized with a parent screen for every set of child screens.

➜ 2.7.1 Back-Button Navigation

Back-button navigation—navigation back through the history of screens—is deeply rooted in the Android system.

The Back button in the bottom left corner of every screen to take them to the previous screen.

The set of historical screens always starts with the user's Launcher (the device's Home screen), as shown in the figure below.

Pressing Back enough times should return the user back to the Launcher.

In the Fig. 2.7.1.

1. Starting from Launcher.
2. Clicking the Back button to navigate to the previous screen.

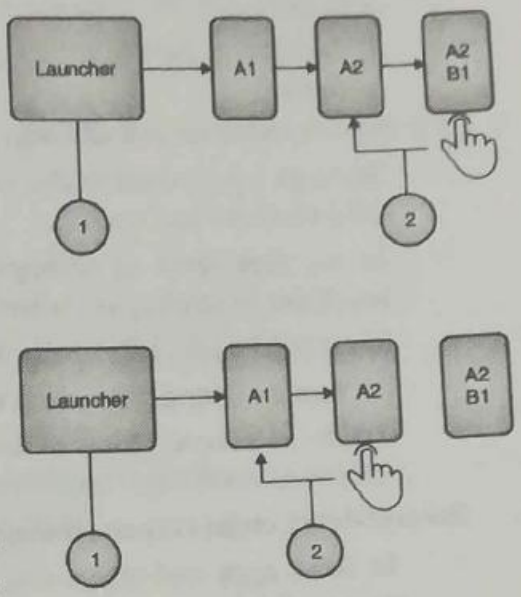


Fig. 2.7.1

- You don't have to manage the Back button in your app. The system handles tasks and the back stack—the list of previous screens—automatically.
- The Back button by default simply traverses this list of screens, removing the current screen from the list as the user presses it.
- You may wish to trigger the embedded browser's default back behaviour when users press the device's Back button.
- The onBackPressed() method of the Activity class is called whenever the activity detects the user's press of the Back key. The default implementation simply finishes the current activity, but you can override this to do something else:

```
@Override
public void onBackPressed() {
  // Add the Back key handler here.
  return;
}
```

- If your code triggers an embedded browser with its own behavior for the Back key, you should return the Back key behavior to the system's default behavior if the user uses the Back key to go beyond the beginning of the browser's internal history.

2.7.2 Hierarchical Navigation Patterns

- To give the user a path through the full range of an app's screens, the best practice is to use some form of hierarchical navigation. An app's screens are typically organized in a parent-child hierarchy, as shown in the Fig. 2.7.2.

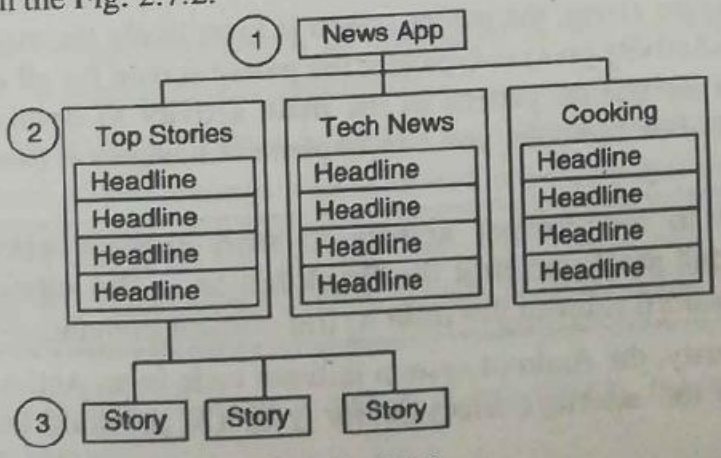


Fig. 2.7.2

In the Fig. 2.7.2.

1. **Parent screen**
   - A parent screen (such as a news app's home screen) enables navigation down to child screens.
   - The main activity of an app is usually the parent screen.
   - Implement a parent screen as an Activity with descendant navigation to one or more child screens.
2. **First-level child screen siblings**
   - Siblings are screens in the same position in the hierarchy that share the same parent screen (like brothers and sisters).
   - In the first level of siblings, the child screens may be collection screens that collect the headlines of stories, as shown above.
   - Implement each child screen as an Activity or Fragment.
   - Implement lateral navigation to navigate from one sibling to another on the same level.
   - If there is a second level of screens, the first level child screen is the parent to the second level child screen siblings. Implement descendant navigation to the second-level child screens.
3. **Second-level child screen siblings**
   - In news apps and others that offer multiple levels of information, the second level of child screen siblings might offer content, such as stories.
   - Implement a second-level child screen sibling as another Activity or Fragment.
   - Stories at this level may include embedded story elements such as videos, maps, and comments, which might be implemented as fragments.

- You can enable the user to navigate up to and down from a parent, and sideways among siblings:
  1. **Descendant navigation :** Navigating down from a parent screen to a child screen.
  2. **Ancestral navigation :** Navigating up from a child screen to a parent screen.
  3. **Lateral navigation :** Navigating from one sibling to another sibling (at the same level).
- You can use a main activity (as a parent screen) and then other activities or fragments to implement a hierarchy of screens within an app.

☞ **Main activity with other activities**

- If the first-level child screen siblings have another level of child screens under them, you should implement the first-level screens as activities, so that their lifecycles are managed properly before calling any second-level child screens.
- For example, in the figure above, the parent screen is most likely the main activity. An app's main activity (usually MainActivity.java) is typically the parent screen for all other screens in your app, and you implement a navigation pattern in the main activity to enable the user to go to other activities or fragments. For example, you can implement navigation using an Intent that starts an activity.

> **Note :** Using an Intent in the current activity to start another activity adds the current activity to the call stack, so that the **Back** button in the other activity (described in the previous section) returns the user to the current activity.

- As you learned previously, the Android system initiates code in an Activity instance with callback methods that manage the activity's lifecycle for you. (A previous lesson covers the activity

lifecycle; for more information, see "Managing the Activity Lifecycle" in the Training section of the Android Developer Develop guide.)

The hierarchy of parent and child activities is defined in the AndroidManifest.xml file. For example, the following defines OrderActivity as a child of the parent MainActivity:

```
<activity android:name=".OrderActivity"
android:label="@string/title_activity_order"
android:parentActivityName=
        "com.example.android.droideafe.MainActivity">

<meta-data
android:name="android.support.PARENT_ACTIVITY"
android:value=".MainActivity"/>
</activity>
```

- **Ancestral navigation (the Up button)**
- With ancestral navigation in a multitier hierarchy, you enable the user to go up from a section sibling to the collection sibling, and then up to the parent screen.

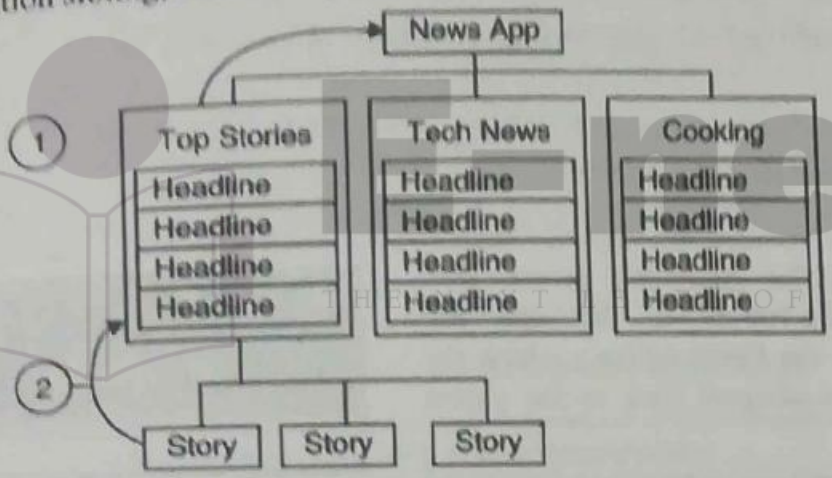


Fig. 2.7.3

In the Fig. 2.7.3,

1. Up button for ancestral navigation from the first-level siblings to the parent.
2. Up button for ancestral navigation from second-level siblings to the first-level child screen acting as a parent screen.

- The **Up** button is used to navigate within an app based on the hierarchical relationships between screens. For example (referring to the figure above):
  - If a first-level child screen offers headlines to navigate to second-level child screens, the second-level child screen siblings should offer **Up** buttons that return to the first-level child screen, which is their shared parent.
  - If the parent screen offers navigation to first-level child siblings, then the first-level child siblings should offer an **Up** button that returns to the parent screen.
  - If the parent screen is the topmost screen in an app (that is, the app's home screen), it should not offer an **Up** button.

Note : The **Back** button below the screen differs from the **Up** button. The **Back** button provides navigation to whatever screen you viewed previously. If you have several children screens that the user can navigate through using a lateral navigation pattern (as described later in this chapter), the **Back** button would send the user back to the previous child screen, not to the parent screen. Use an **Up** button if you want to provide ancestral navigation from a child screen back to the parent screen..

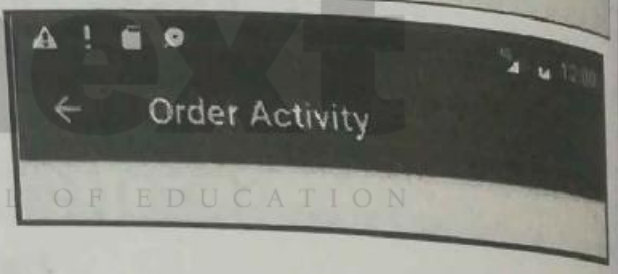
- To provide the **Up** button for a child screen activity, declare the activity's parent to be MainActivity in the AndroidManifest.xml file. You can also set the android:label to a title for the activity screen, such as "Order Activity" (extracted into the string resource title_activity_order in the code below). Follow these steps to declare the parent in AndroidManifest.xml:

☞ Open **AndroidManifest.xml**.

- Change the activity element for the child screen activity (in this example, OrderActivity) to the following:

```
<activity android:name=".OrderActivity"
android:label="@string/title_activity_order"
android:parentActivityName=
    "com.example.android.optionsmenuorderactivity.MainActivity">
<meta-data
  android:name="android.support.PARENT_ACTIVITY"
  android:value=".MainActivity"/>
</activity>
```

- The child ("Order Activity") screen now includes the **Up** button in the app bar (highlighted in the figure below), which the user can tap to navigate back to the parent screen.



☞ **Descendant navigation**

- With descendant navigation, you enable the user to go from the parent screen to a first-level child screen, and from a first-level child screen down to a second-level child screen.

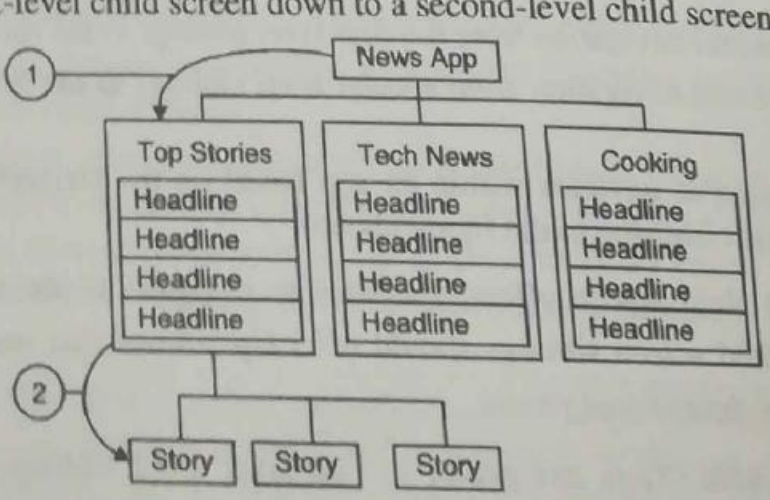


Fig. 2.7.4

In the Fig. 2.7.4.

1. Descendant navigation from parent to first-level child screen.

Descendant navigation from headline in a first-level child screen to a second-level child screen.

Buttons or targets

The best practice for descendant navigation from the parent screen to collection siblings is to use buttons or simple targets such as an arrangement of images or iconic buttons (also known as a dashboard). When the user touches a button, the collection sibling screen opens, replacing the current context (screen) entirely.

Note : Buttons and simple targets are rarely used for navigating to section siblings within a collection.

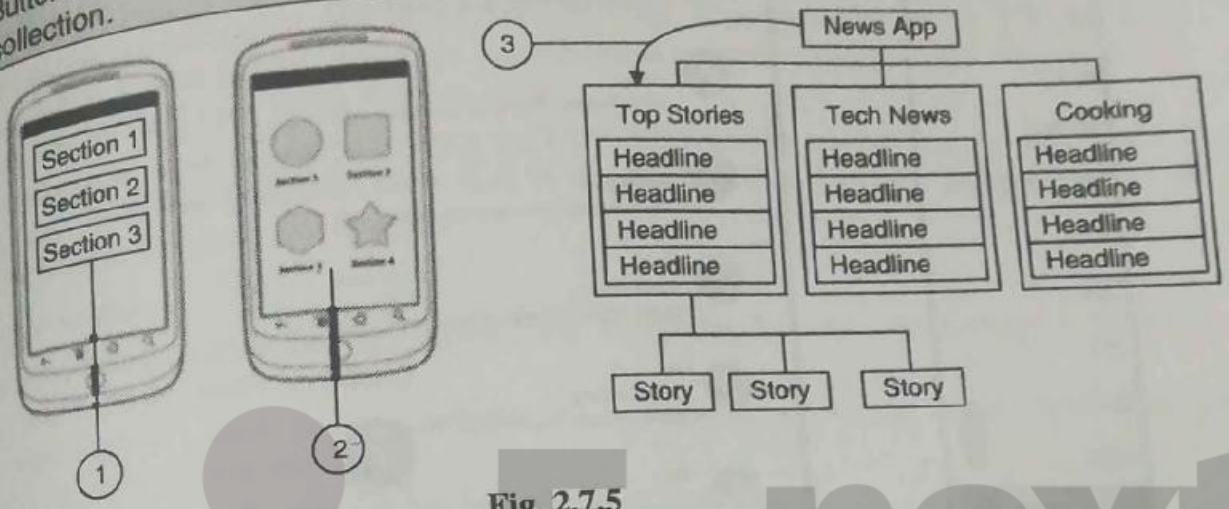


Fig. 2.7.5

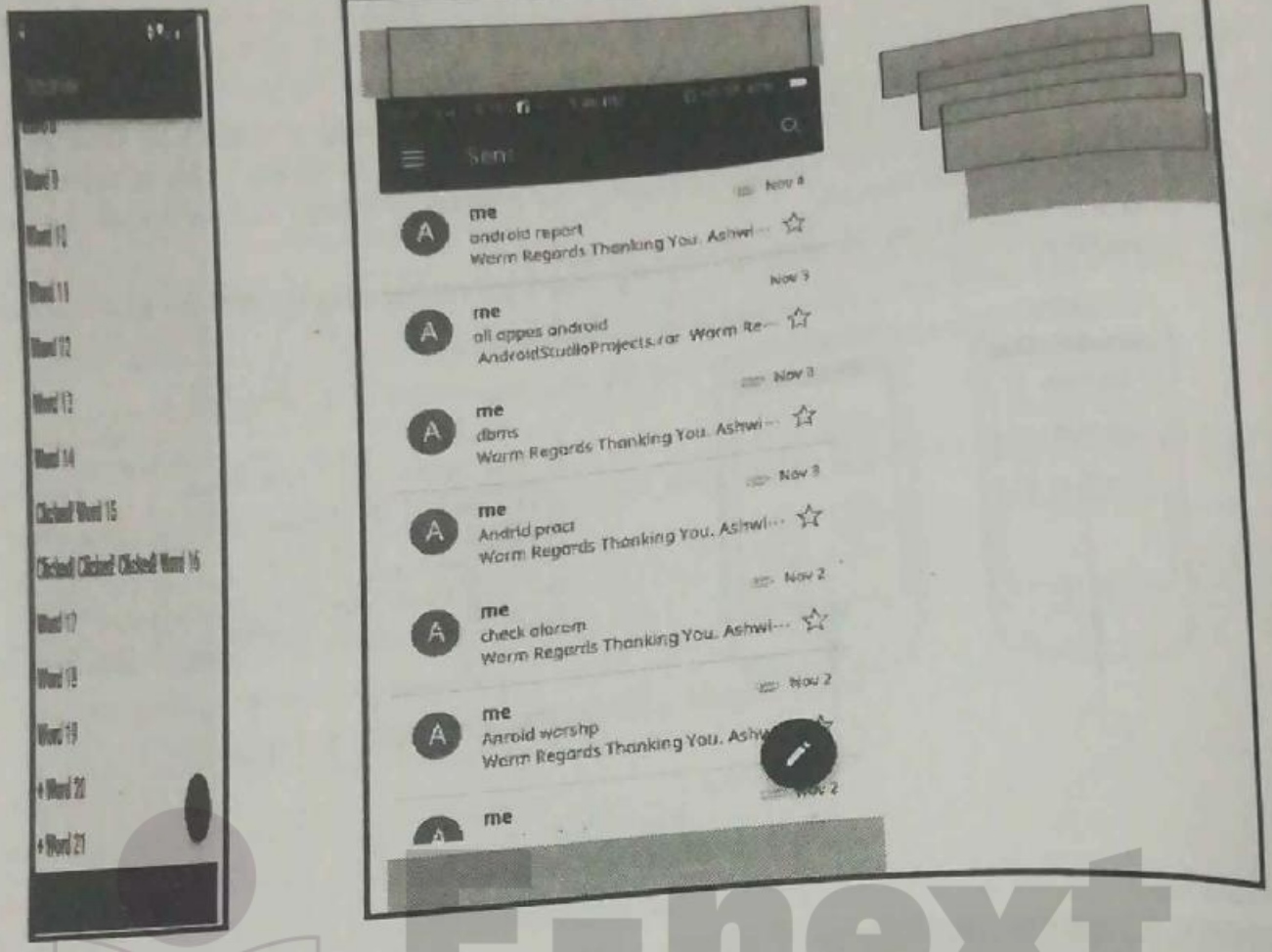
In the Fig. 2.7.6.

1. Buttons on a parent screen.
2. Targets (Image buttons or icons) on a parent screen.
3. Descendant navigation pattern from parent screen to first-level child siblings.

Syllabus Topic : RecyclerView

2.8 RecyclerView

- When you display a large number of items in a scrollable list, most items are not visible. For example, in a long list of words or many news headlines, the user only sees a small number of list items at a time.
- The RecyclerView class is a more advanced and flexible version of ListView. It is a container for displaying large data sets that can be scrolled very efficiently by maintaining a limited number of views.
- Use the RecyclerView widget when display a large amount of scrollable data, or data collections whose elements change at runtime based on user action or network events.



RecyclerView components

– To display your data in a RecyclerView, you need the following parts:

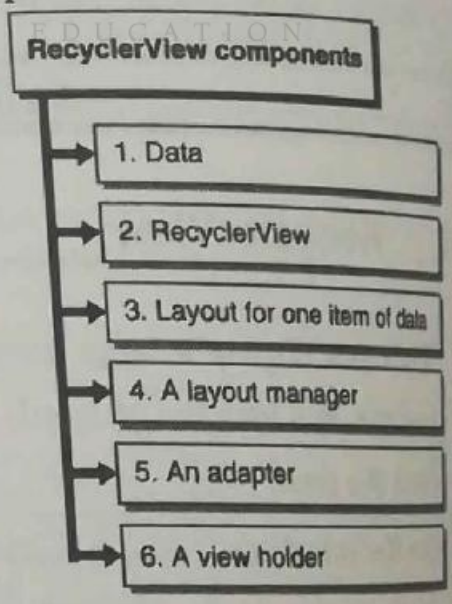


Fig. C.2.5 :Components of RecylerView

➜ **1. Data**

It doesn't matter where the data comes from. You can create the data locally, as you do in the practical, get it from a database on the device as you will do in a later practical, or pull it from the cloud.

➜ **2. A RecyclerView**

– The scrolling list that contains the list items.

– An instance of RecyclerView as defined in your activity's layout file to act as the container for the views.

➜ **3. Layout for one item of data**

All list items look the same, so you can use the same layout for all of them. The item layout has to be created separately from the activity's layout, so that one item view at a time can be created and filled with data.

4. A layout manager

- The layout manager handles the organization (layout) of user interface components in a view.
- All view groups have layout managers. For the LinearLayout, the Android system handles the layout for you. RecyclerView requires an explicit layout manager to manage the arrangement of list items contained within it. This layout could be vertical, horizontal, or a grid.
- The layout manager is an instance of Recyclerview.LayoutManager to organize the layout of the items in the RecyclerView.

5. An adapter

- The adapter connects your data to the RecyclerView. It prepares the data and how will be displayed in a view holder. When the data changes, the adapter updates the contents of the respective list item view in the RecyclerView.
- And an adapter is an extension of RecyclerView.Adapter. The adapter uses a ViewHolder to hold the views that constitute each item in the RecyclerView, and to bind the data to be displayed into the views that display it.

6. A view holder

- The view holder extends the ViewHolder class. It contains the view information for displaying one item from the item's layout.
- A view holder used by the adapter to supply data, which is an extension of RecyclerView.ViewHolder
- The Fig. 2.8.1 shows the relationship between these components.

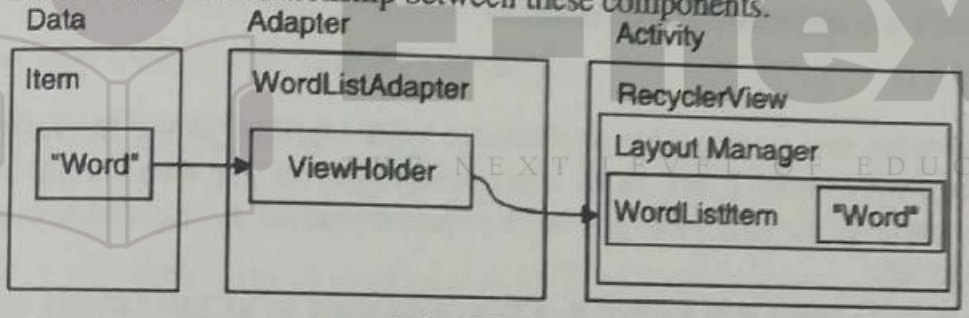


Fig. 2.8.1

- Any displayable data can be shown in a RecyclerView.
  1. Text 2. Images 3. Icons
- Data can come from any source.
  1. Created by the app. For example, scrambled words for a game.
  2. From a local database. For example, a list of contacts.
  3. From cloud storage or the internet. For example news headlines.

RecyclerView

- A View group for a scrollable container
- Ideal for long lists of similar items
- Uses only a limited number of views that are re-used when they go off-screen. This saves memory and makes it faster to update list items as the user scrolls through data, because it is not necessary to create a new view for every item that appears.
- In general, the RecyclerView keeps as many item views as fit on the screen, plus a few extra at each end of the list to make sure that scrolling is fast and smooth.

☞ **Item Layout**

- The layout for a list item is kept in a separate file so that the adapter can create item views and fill their contents independently from the layout of the activity.

☞ **Layout Manager**

- A layout manager positions item views inside a view group, such as the RecyclerView and determines when to reuse item views that are no longer visible to the user.
- To reuse (or recycle) a view, a layout manager may ask the adapter to replace the contents of the view with a different element from the dataset. Recycling views in this manner improves performance by avoiding the creation of unnecessary views or performing expensive findViewById() lookups.
- RecyclerView provides these built-in layout managers:
- LinearLayoutManager shows items in a vertical or horizontal scrolling list.
- GridLayoutManager shows items in a grid.
- StaggeredGridLayoutManager shows items in a staggered grid.
- To create a custom layout manager, extend the RecyclerView.LayoutManager class.

2.8.1 Animations

- Animations for adding and removing items are enabled by default in RecyclerView. To customize these animations, extend the RecyclerView.ItemAnimator class and use the RecyclerView.setItemAnimator()method.

☞ **Adapter**

- An Adapter helps two incompatible interfaces to work together. In the RecyclerView, the adapter connects data with views. It acts as an intermediary between the data and the view. The Adapter receives or retrieves the data, does any work required to make it displayable in a view, and places the data in a view.
- For example, the adapter may receive data from a database as a Cursor object, extract the the word and its definition, convert them to strings, and place the strings in an item view that has two text views, one for the word and one for the definition. You will learn more about cursors in a later chapter.
- The RecyclerView.Adapter implements a view holder, and must override the following callbacks:
- onCreateViewHolder() inflates an item view and returns a new view holder that contains it. This method is called when the RecyclerView needs a new view holder to represent and item.
- onBindViewHolder() sets the contents of an item at a given position in the RecyclerView. This is called by the RecyclerView, for example, when a new item scrolls into view.

☞ **View holder**

- A RecyclerView.ViewHolder describes an item view and metadata about its place within the RecyclerView. Each view holder holds one set of data. The adapter adds data to view holders for the layout manager to display.
- You define your view holder layout in an XML resource file. It can contain (almost) any type of view, including clickable elements.

Implementing a Recycler View

Implementing a RecyclerView requires the following steps:

1. Add the RecyclerView dependency to the app's app/build.gradle file.
2. Add the RecyclerView to the activity's layout
3. Create a layout XML file for one item
4. Extend RecyclerView.Adapter and implement onCrateViewHolder and onBindViewHolder methods.
5. Extend RecyclerView.ViewHolder to create a view holder for your item layout. You can add click behavior by overriding the onClick method.
6. In your activity, In the onCreate method, create a RecyclerView and initialize it with the adapter and a layout manager.

Add the dependency to app/build.gradle

Add the recycler view library to your app/build.gradle file as a dependency. Look at the chapter on support libraries or the RecyclerView practical, if you need detailed instructions.

```
dependencies {
  ...
  compile 'com.android.support:recyclerview-v7:24.1.1'
  ...
}
```

Add a RecyclerView to your activity's layout

Add the RecyclerView in your activity's layout file.

```
<android.support.v7.widget.RecyclerView
  android:id="@+id/recyclerview"
  android:layout_width="match_parent"
  android:layout_height="match_parent">
</android.support.v7.widget.RecyclerView>
```

Use the recycler view from the support library to be compatible with older devices. The only required attributes are the id, along with the width and height. Customize the items, not this view group.

Create the layout for one item

Create an XML resource file and specify the layout of one item. This will be used by the adapter to create the view holder.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:orientation="vertical"
  android:padding="6dp">

  <TextView
    android:id="@+id/word"
```

```
    style="@style/word_title" />
</LinearLayout>
```

- The text view has a @style element. A style is a collection of properties that specifies the look of a view. You can use styles to share display attributes with multiple views. An easy way to create a style is to extract the style of a UI element that you already created.
- For example, after styling a TextView, **Right-click > Refactor > Extract > Style** on the element and follow the dialog prompts. More details on styles are in the practical and in a later chapter.

☞ **Create an adapter with a view holder**

- Extend RecyclerView.Adapter and implement the onCrateViewHolder and onBindViewHolder methods.
- Create a new Java class with the following signature:

```
public class WordListAdapter extends RecyclerView.Adapter<WordListAdapter.WordViewHolder>
{}
```

- In the constructor, get an inflater from the current contex, and your data.

```
public WordListAdapter(Context context, LinkedList<String> wordList) {
    mInflater = LayoutInflater.from(context);
    this.mWordList = wordList;
}
```

- For this adapter, you have to implement 3 methods.
- onCreateViewHolder() creates a view and returns it.

```
@Override
public WordViewHolder onCreateViewHolder(ViewGroup parent, int viewType){
    // Inflate an item view.
    View mItemView = mInflater.inflate(R.layout.wordlist_item, parent, false);
    return new WordViewHolder(mItemView, this);
}
```

- onBindViewHolder() associates the data with the view holder for a given position in the RecyclerView.

```
@Override
public void onBindViewHolder(WordViewHolder holder, int position) {
    // Retrieve the data for that position
    String mCurrent = mWordList.get(position);
    // Add the data to the view
    holder.wordItemView.setText(mCurrent);
}
```

- getItemCount() returns to number of data items available for displaying.

```
@Override
public int getItemCount() {
    return mWordList.size();
}
```

- Implement the view holder class
- Extend RecyclerView.ViewHolder to create a view holder for your item layout. You can add click behavior by overriding the onClick method.
- This class is usually defined as an inner class to the adapter and extends RecyclerView.ViewHolder.
- class WordViewHolder extends RecyclerView.ViewHolder { }
- If you want to add click handling, you need to implement a click listener. One way to do this is to have the view holder implement the click listener methods.
- // Extend the signature of WordVewHolder to implement a click listener.
- class WordViewHolder extends RecyclerView.ViewHolder implements View.OnClickListener { }
- In its constructor, the view holder has to inflate its layout, associate with its adapter, and, if applicable, set a click listener.

```
public WordViewHolder(View itemView, WordListAdapter adapter) {
  super(itemView);
  wordItemView = (TextView) itemView.findViewById(R.id.word);
  this.mAdapter = adapter;
  itemView.setOnClickListener(this);
}
```

- And, if you implementing onClickListener, you also have to implement onClick().

```
@Override
public void onClick(View v) {
  wordItemView.setText ("Clicked! "+ wordItemView.getText());
}
```

- Note that to attach click listeners to other elements of the view holder, you do that dynamically in onBindViewHolder.
- Finally, to tie it all together, in your activity's onCreate() method:

☞ **Get a handle to the RecyclerView.**

```
mRecyclerView = (RecyclerView) findViewById(R.id.recyclerview);
```

- Create an adapter and supply the data to be displayed.

```
mAdapter = new WordListAdapter(this, mWordList);
```

- Connect the adapter with the recycler view.

```
mRecyclerView.setAdapter(mAdapter);
```

- Give the recycler view a default layout manager.

```
mRecyclerView.setLayoutManager(new LinearLayoutManager(this));
```

- RecyclerView is an efficient way for displaying scrolling list data. It uses the adapter pattern to connect data with list item views. To implement a RecyclerView you need to create an adapter and a view holder, and the methods that take the data and add it to the list items.

Review Questions

Q. 1 Explain the attributes of input control. **(Refer section 2.1)**

Q. 2 How to design flat buttons ? **(Refer section 2.1.2)**

Q. 3 How to design image button? Explain it. **(Refer section 2.1.3)**

Q. 4 Explain callback methods of Listener. **(Refer section 2.1.5)**

Q. 5 Explain different types of menus of Android. **(Refer section 2.6)**

Q. 6 Explain how to create options menu in Android ? **(Refer section 2.6.2(A))**

Q. 7 Explain how to create context menu in Android ? **(Refer section 2.6.2(B))**

❑❑❑