DAYANANDA SAGAR COLLEGE OF ENGG.

(An Autonomous Institute Affiliated to Visvesvaraya Technological University, Belagavi & Approved by AICTE, New Delhi.)

Department of Computer Science And Engg.

Subject: Mobile Application Development(MAD)

By

Dr. Rohini T V,

Associate Professor

V: A & B Section

Department of Computer Science & Engineering

Aca. Year ODD SEM /2022-23

USER INTERACTION(Module 2)

Students will be learning how to design user interaction ,User Experience and Test User Interaction

Topic Name

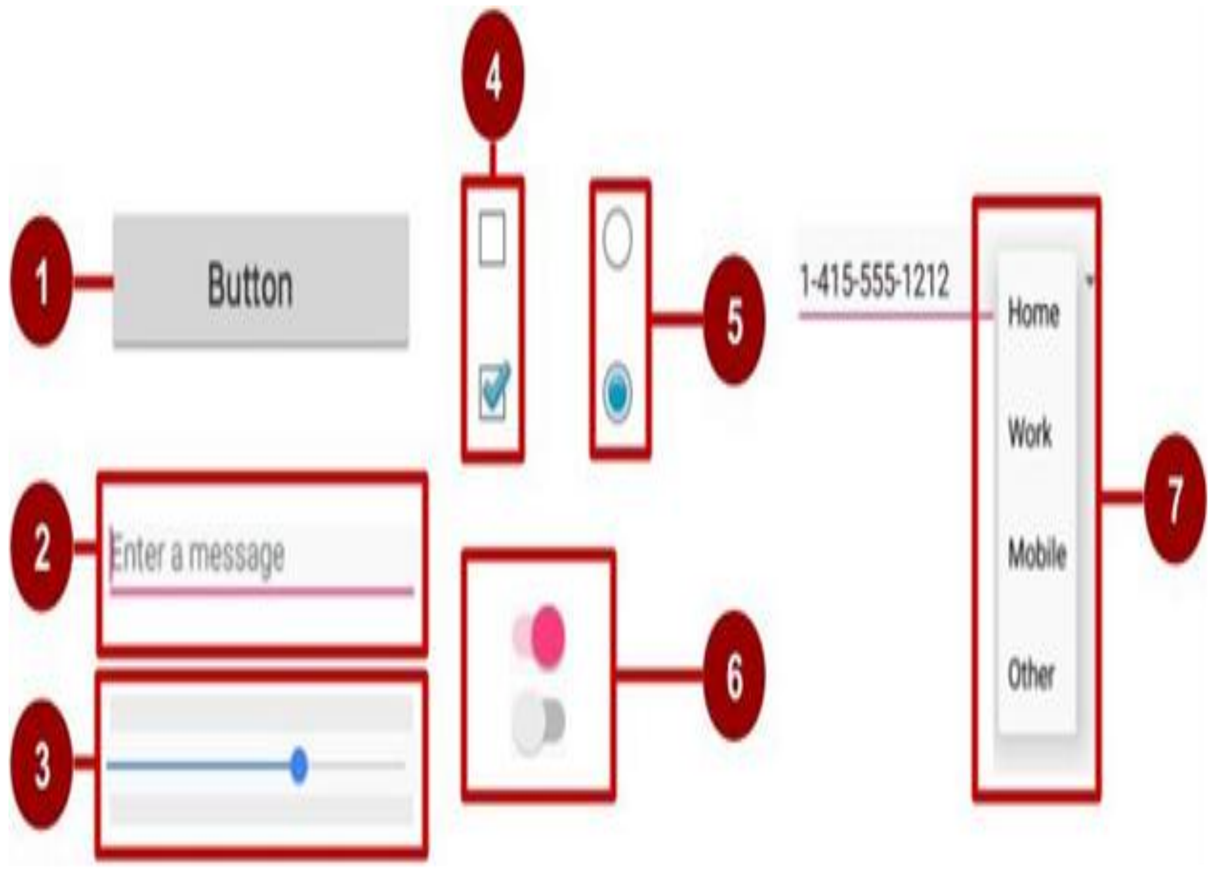
- User Interaction
- Delightful User experience
- Testing your UI

User Input Controls

- Interaction design for user input
- Input controls and view focus
- Using buttons
- Using input controls for making choices
- Text input
- Using dialogs and pickers
- Recognizing gestures

Interaction design for user input

- The reason you design an app is to provide some function to a user, and in order to use it, there must be a way for the user to interact with it.
- For an Android app, interaction typically 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, and a microphone.



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

Input controls and view focus

- Android applies 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. In previous chapters, we have learned that View is the base 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? For example, if you have several TextView objects and an EditText object in your app, which UI component (that is, which View) receives text typed by the user first?

- The View that "has the focus" will be the component that receives user input.
- Focus indicates which view is currently selected to receive input. Focus can be initiated by the user by touching a View,such as a TextView or an EditText object. You can define a focus order in which the user is guided from UI control to UI control using the Return key, Tab key, or arrow keys. Focus can also be programmatically controlled; a programmer can requestFocus() on any View that is focusable.

- Focus movement is based on an algorithm that finds the nearest neighbor in a given direction:
- When the user touches the screen, the topmost view under the touch is in focus, providing touch-access for the child views of the topmost view.
- If you set an EditText view to a single-line, the user can tap the Return key on the keyboard to close the keyboard and shift focus to the next input control view based on what the Android system finds:
- The system usually finds the nearest input control in the same direction the user was navigating (up, down, left, or right).
- If there are multiple input controls that are nearby and in the same direction, the system scans from left to right, top to bottom.
- Focus can also shift to a different view if the user interacts with a directional control, such as a directional pad (d-pad) or trackball

If the algorithm does not give you what you want, you can override it by adding the nextFocusDown , nextFocusLeft ,

nextFocusRight , and nextFocusUp XML attributes to your layout file.

1. Add 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.

2. Define the value of the attribute to be the id of the next view. For example:

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

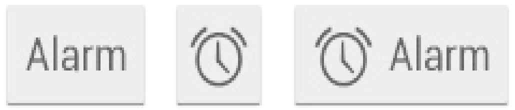
Using buttons

- People like to press buttons. Show someone a big red button with a message that says "Do not press" and the person will likely press it for the sheer pleasure of pressing a big red button (that the button is forbidden is also a factor).

You can make a Button using:

- Only text, as shown on the left side of the figure below.
- Only an icon, as shown in the center of the figure below.
- Both text and an icon, as shown on the right side of the figure below.

When touched or clicked, a button performs an action. The text and/or icon provides a hint of that action. It is also referred to as a "push-button" in Android documentation



Creating a raised button with an icon and text

While a button usually displays text that tells the user what the button is for, raised buttons can also display icons along with text.

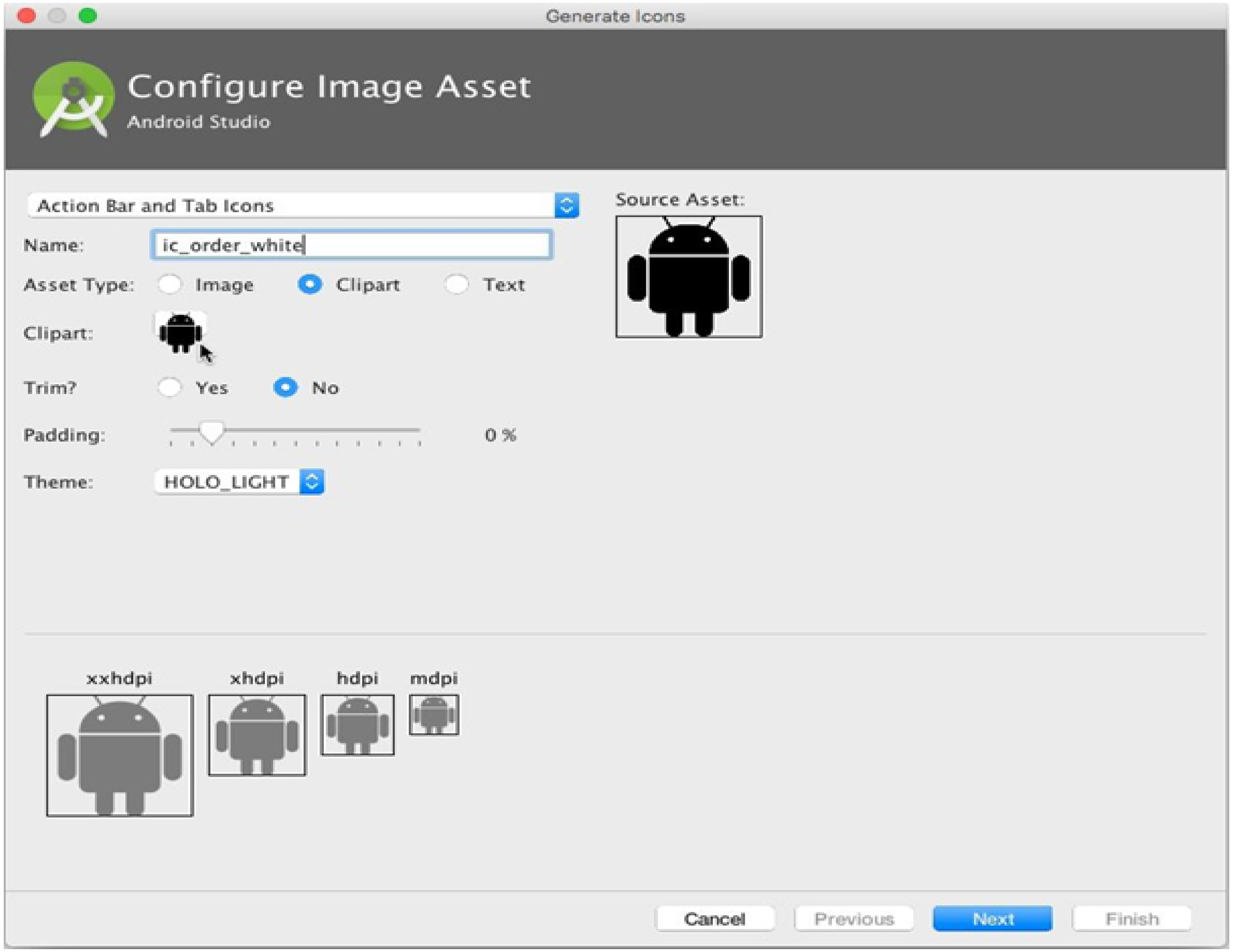
Choosing an icon

To choose images of a standard icon that are resized for different displays, follow these steps:

1. Expand app > res in the Project view, and right-click (or Command-click) drawable.

2. Choose New > Image Asset. The Configure Image Asset dialog appears. .
3.Choose Action Bar and Tab Items in the drop-down menu of the Configure Image Asset dialog (see Image Asset Studio for a complete description of this dialog.)

4. Click the Clipart: image (the Android logo) to select a clipart image as the icon. A page of icons appears as shown below. Click the icon you want to use.


Generate Icons

Configure Image Asset
Android Studio

Action Bar and Tab Icons

Name: ic_order_white

Asset Type: Image [✓ Clipart] Text

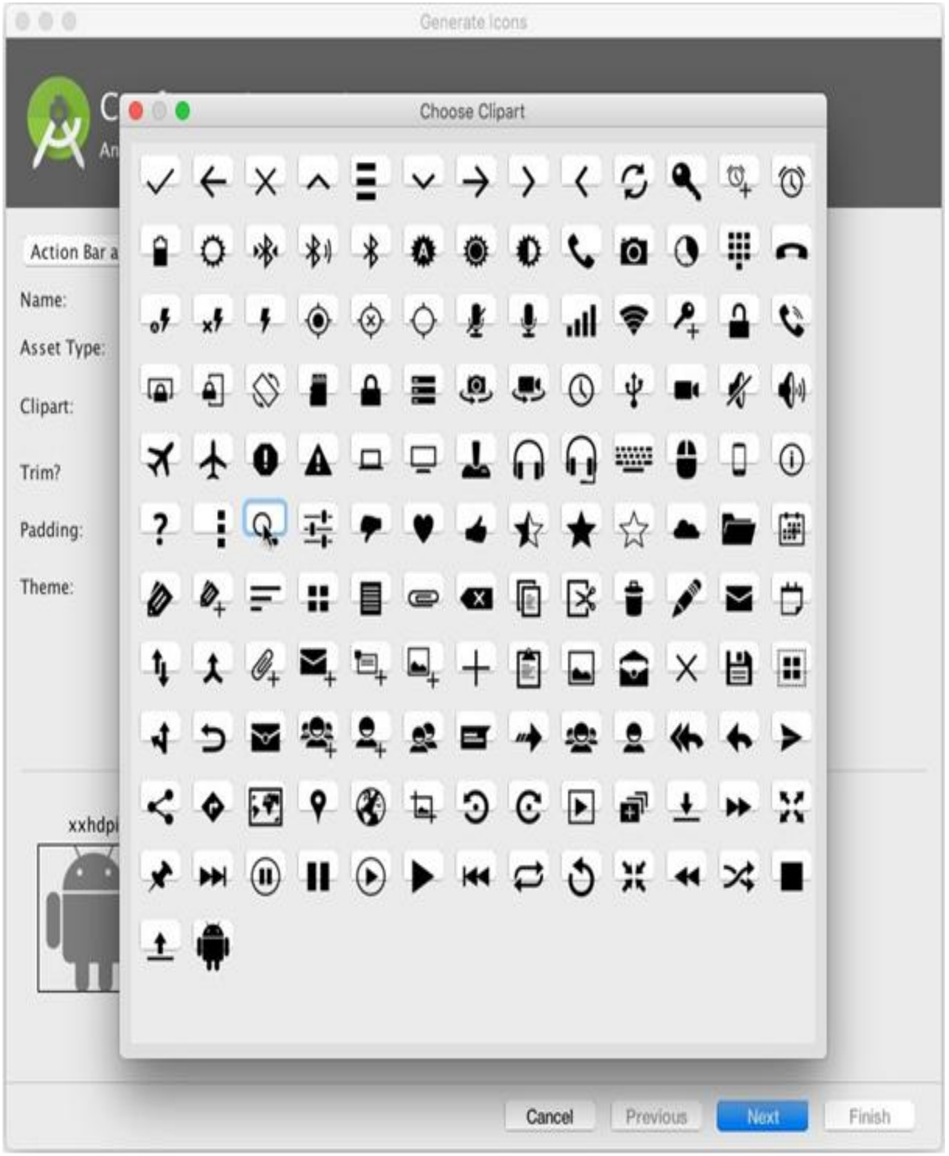
Clipart:

Trim? Yes [✓ No]

Padding: 0 %

Theme: HOLO_LIGHT

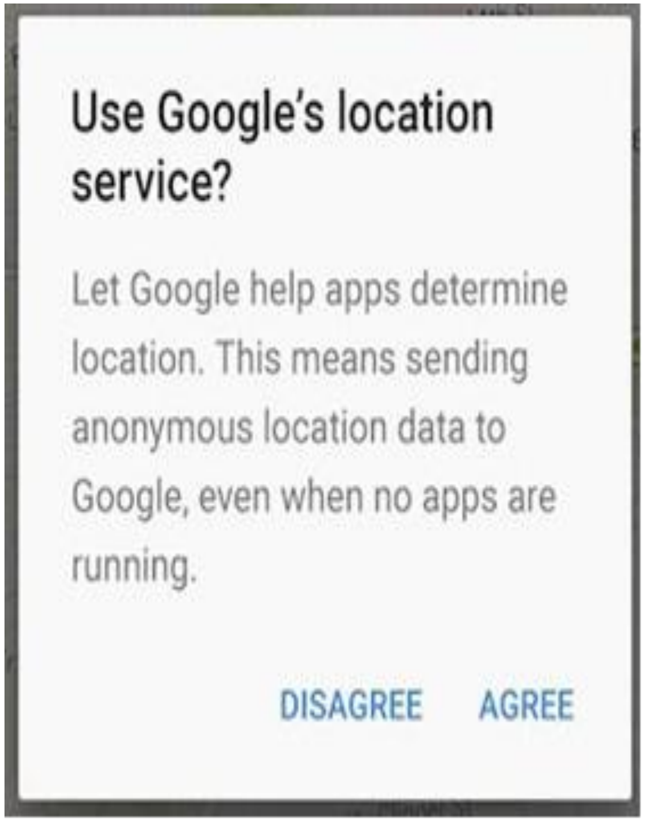
Source Asset:



5. You may want to make the following adjustments:
Choose HOLO_DARK from the Theme drop-down menu to sets the icon to be white against a dark-colored (or black) background.
Depending on the shape of the icon, you may want to add padding to the icon so that the icon doesn't crowd the text. Drag the Padding slider to the right to add more padding.
6. Click Next, and then click Finish in the Confirm Icon Path dialog. The icon name should now appear in the app > res
> drawable folder.

Designing flat buttons :

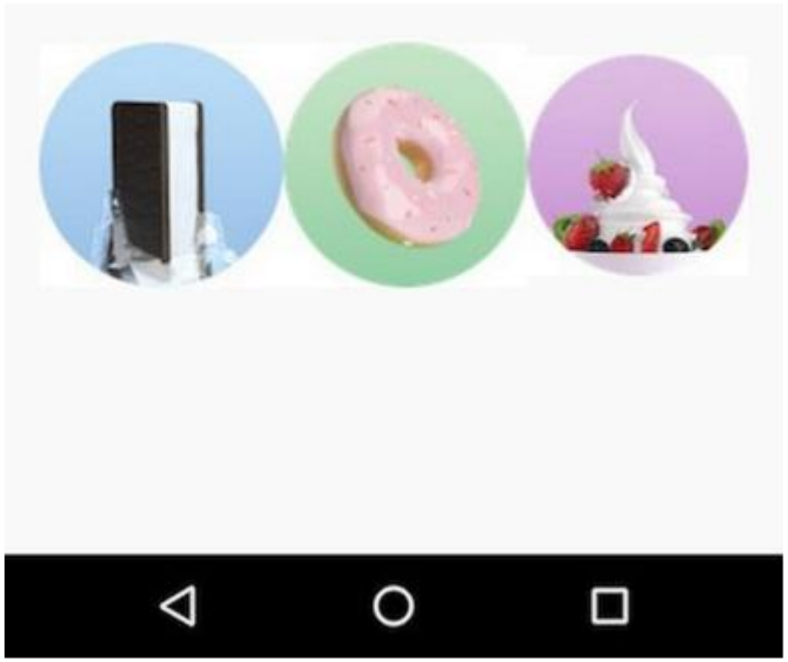
- A flat button, also known as a borderless button, is a text-only button that appears flat on the screen without a shadow.
- The major benefit of flat buttons is simplicity — they minimize distraction from content. Flat buttons are useful when you have a dialog, as shown in the figure below, which requires user input or interaction.
- In this case, 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 button.


Use Google's location service?

Let Google help apps determine location. This means sending anonymous location data to Google, even when no apps are running.

Designing images as buttons

- You can turn any View, such as an ImageView, into a button 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.
- Note: To bring images into your Android Studio project, create or save the image in JPEG format, and copy the image file into the app > src > main > res > drawables folder of your project. For more information about drawable resources, see Drawable Resources in the App Resources section of the Android Developer Guide.
- If you are using multiple images as buttons, arrange them in a viewgroup so that they 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:



```
<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>
```

Responding to button-click events

- Use an *event listener called OnClickListener, which is an interface in the View class, to respond to the click event that occurs when the user taps or clicks a clickable object, such as a Button, ImageButton, or FloatingActionButton .*
- **Adding onClick to the layout element**

To set up *an OnClickListener* for the clickable object in your Activity code and assign a callback method, use the *android:onClick attribute* with the clickable object's element in the XML layout. For 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" />
```

- when a user clicks the button, the Android system calls the Activity's *sendMessage()* method:

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

Using the button-listener design pattern

- Use the event listener *View.OnClickListener*, which is an interface in the View class that contains a single callback method, *onClick()*. The method is called by the Android framework when the view is triggered by user interaction.
- The event listener must already be registered to the view in order to be called for the event. Follow these steps to register the listener and use it.

1. 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);

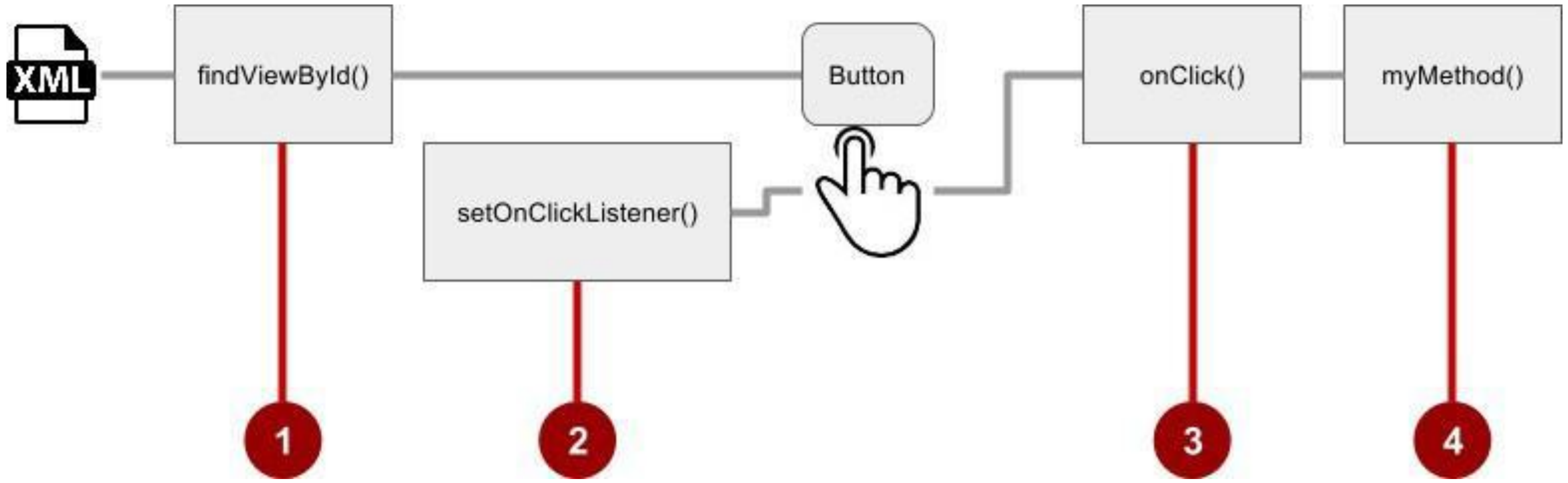
2. Get a new View.OnClickListener object and register it to the button by calling the setOnClickListener() method. The argument to setOnClickListener() takes an object that implements the View.OnClickListener interface, which has one method: onClick() .

   button.setOnClickListener(new View.OnClickListener() {

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

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

4.Create a method to do something in response to the button click, such as perform an action.



To set the click listener programmatically instead of with the onClick attribute, customize the View.OnClickListener class and override its onClick() handler to perform some action, as shown below:

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

Using the event listener interface for other events

The following are some of the listeners available in the Android framework and the callback methods associated with each one:

*1.**onClick()*** from View.OnClickListener: 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()*** from View.OnLongClickListener: 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. That is, return true to indicate that you have handled the event and it should stop here; return false if you have not handled it and/or the event should continue to any other on-click listeners.

*3.**onTouch()*** from View.OnTouchListener: Handles any form of touch contact with the screen including individual or multiple touches and gesture motions, including a press, a release, or any movement gesture on the screen (within the bounds of the UI element). 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()*** 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()*** from View.OnKeyListener: Handles when a key on a hardware device is pressed while a view has focus.

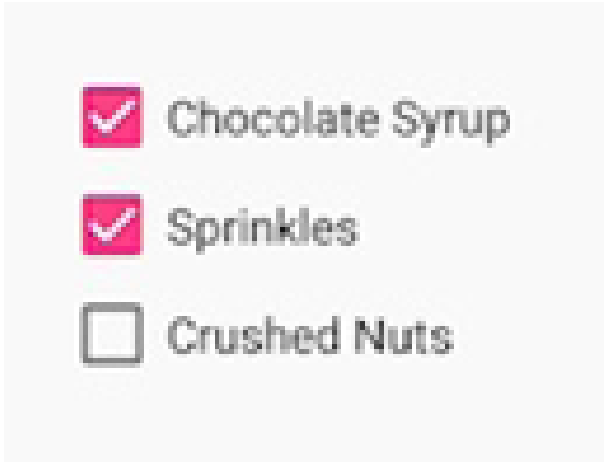
Using input controls for making choices

Android offers ready-made input controls for the user to select one or more choices:

1. Checkboxes: Select one or more values from a set of values by clicking each value's checkbox.
2. 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.
3. Toggle button: Select one state out of two or more states. Toggle buttons usually offer two visible states, such as "on“ and "off".
4. Spinner: Select one value from a set of values in a drop-down menu. Only one value can be selected. Spinners are for three or more choices, and takes up little room in your layout.

Checkboxes

- Use 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. (If you want to limit the user's selection to only one item of a set, use radio buttons.) 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.



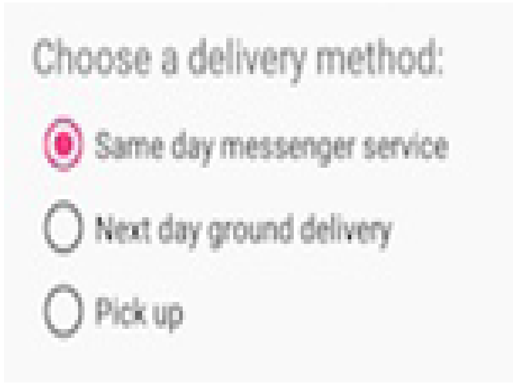
You create each checkbox using a CheckBox element in your XML layout.

To create multiple checkboxes in a vertical orientation, use a vertical LinearLayout:

```
<LinearLayout
xmlns:android="http://schemas.android.co
m/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>
```

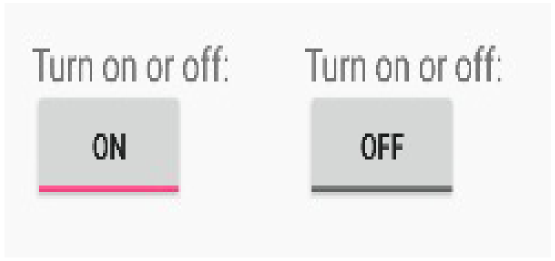
Radio buttons

- Use radio buttons when you have two or more options that are mutually exclusive—the user must select only one of them.
- Users expect radio buttons to appear as a vertical list, or side-by-side horizontally if the labels are short.
- 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:



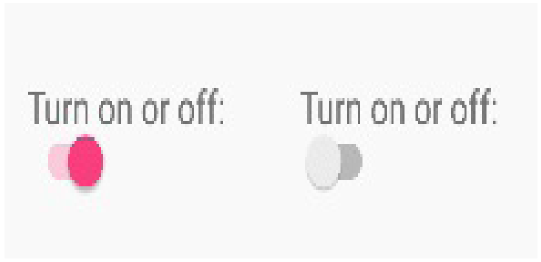
Toggle buttons and switches

A toggle input control lets the user change a setting between two states. Android provides the ToggleButton class, which shows a raised button with "OFF" and "ON".



Examples of 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.



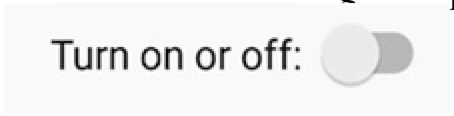
Using a switch

A switch is a separate instance of the Switch class, which extends the CompoundButton class just like ToggleButton.

Create a toggle switch by using a Switch element in your XML layout:

```
<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"/>
```

The android:text attribute defines a string that appears to the left of the switch, as shown below:



To respond to the switch tap, declare an android:onClick callback method for the Switch —the code is basically the same as for a ToggleButton . The method must be defined in the activity hosting the layout, and it must be public , return void , and define a View as its only parameter (this will be the view that was clicked). Use CompoundButton.OnCheckedChangeListener() to detect the state change of the switch

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.

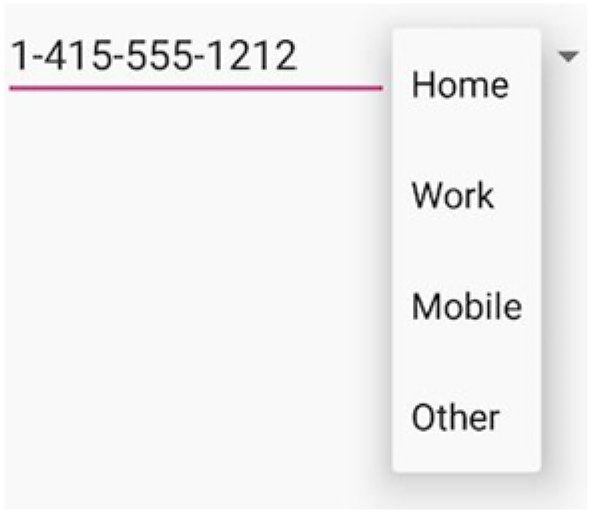
If you have a long list of choices, a spinner may extend beyond your layout, forcing the user to scroll it. A spinner scrolls automatically, with no extra code needed. However, scrolling a long list (such as a list of countries) is not recommended as it can be hard to select an item.

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 and its adapter

Create the spinner, and set its listener to the activity that implements the callback methods. Follow these steps :

1.Add the code below to the onCreate() method, which does the following:

2. Gets the spinner object you added to the layout using findViewById() to find it by its id (label_spinner).

3.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);
    }
}
```

4. 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);

As shown above, you use the createFromResource() method, which takes as arguments:

5. The activity that implements the callbacks for processing the results of the spinner (this)
6. The array (labels_array)
7. The layout for each spinner item (layout.simple_spinner_item).
8. 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, you can use

simple_spinner_dropdown_item as your layout:

9. Use setAdapter() to apply the adapter to the spinner:

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

```
@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);
}
```

Text input

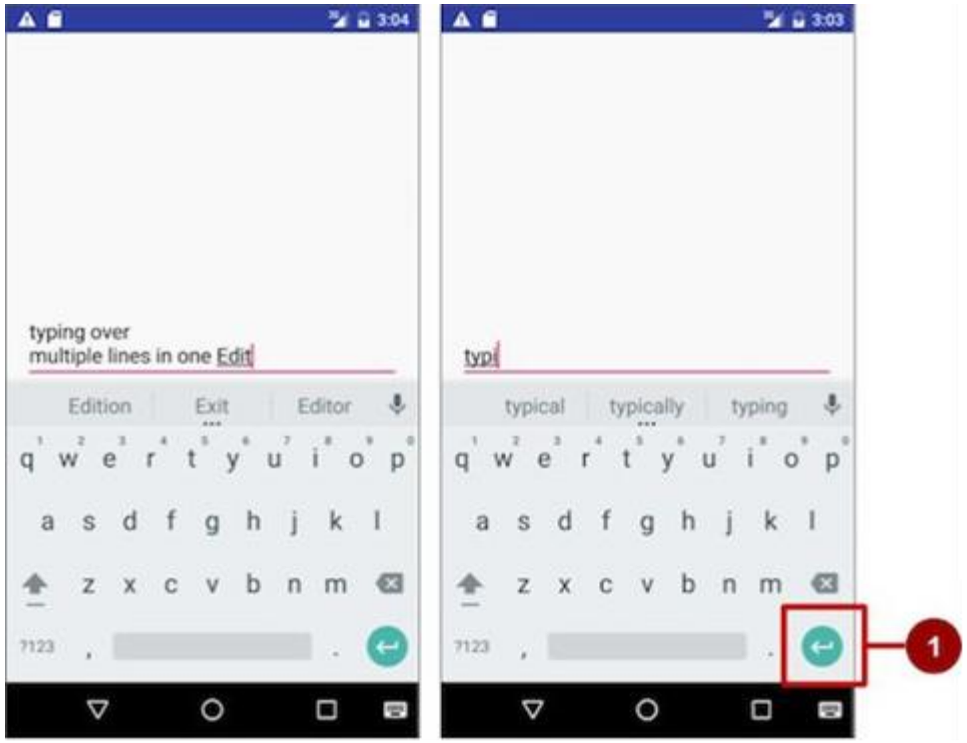
Use the EditText class to get user input that consists of textual characters, including numbers and symbols. EditText extends the TextView class, to make the TextView editable. Customizing an EditText object for user input .In the Layout Manager of Android Studio, create an EditText view by adding an 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>
```

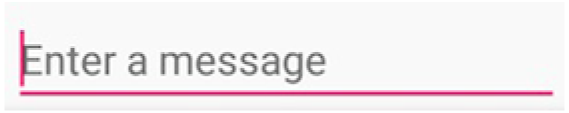
Enabling multiple lines of input

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



Attributes for customizing an EditText view

- 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".



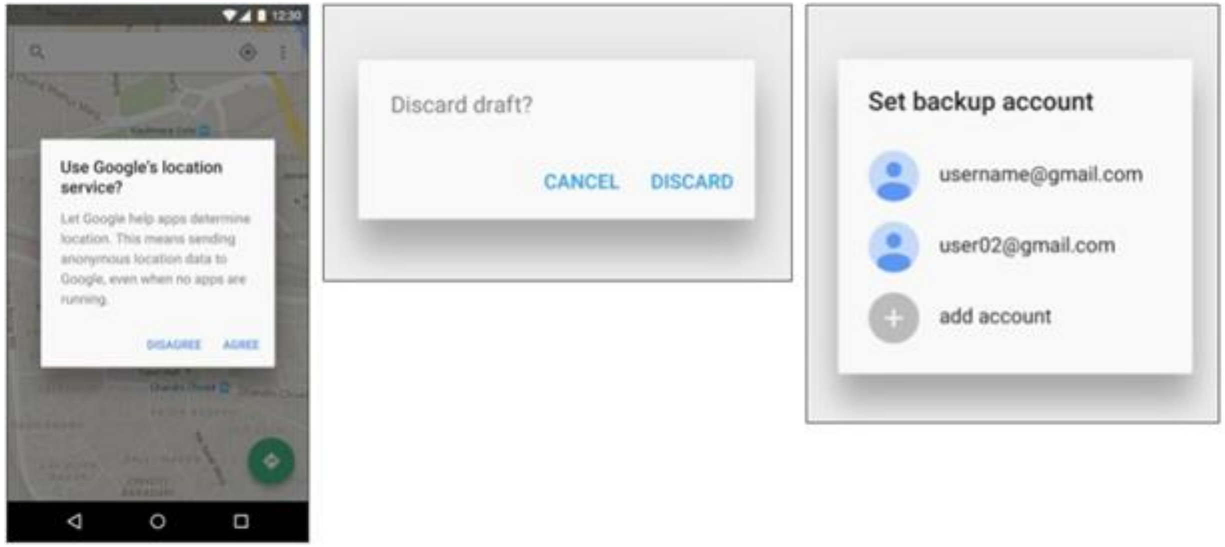
Changing keyboards and input behaviors

The Android system shows an on-screen keyboard—known as a soft input method—when a text field in the UI receives focus. To provide the best user experience, you can specify characteristics about the type of input the app expects, such as whether it's a phone number or email address.

- textCapSentences : 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.
- textEmailAddress : For email entry, show an email keyboard with the "@" symbol conveniently located next to the space key.
- phone : For phone number entry, show a numeric phone keypad.

Using 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.
- For example, you would typically use a dialog to show an alert that requires users to tap a button make a decision, such as OK or Cancel.
- In the figure below, the left side shows an alert with Disagree and Agree buttons, and the center shows an alert with Cancel and Discard buttons. You can also use a dialog to provide choices in the style of radio buttons, as shown on the right side of the figure below


Use Google's location service?

Let Google help apps determine location. This means sending anonymous location data to Google, even when no apps are running.

Discard draft?

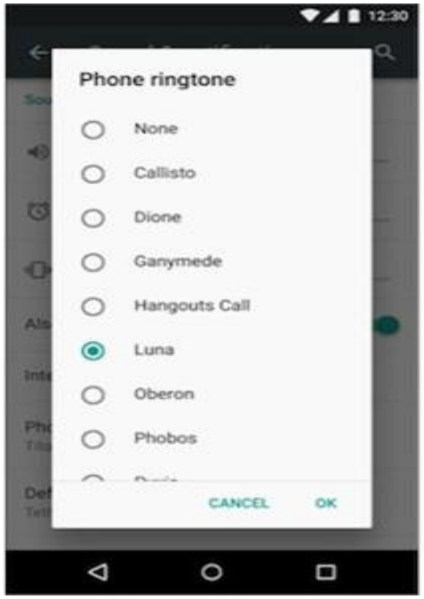
Set backup account

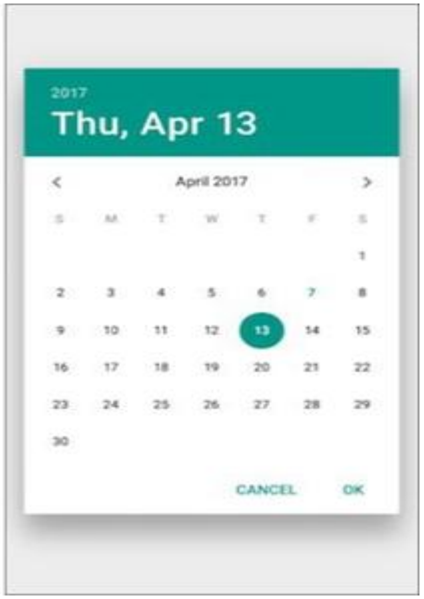
username@gmail.com

user02@gmail.com

add account

- The base class for all dialog components is a Dialog. There are several useful Dialog subclasses for alerting the user on a condition, showing status or progress, displaying information on a secondary device, or selecting or confirming a choice, as shown on the left side of the figure below.
- 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.



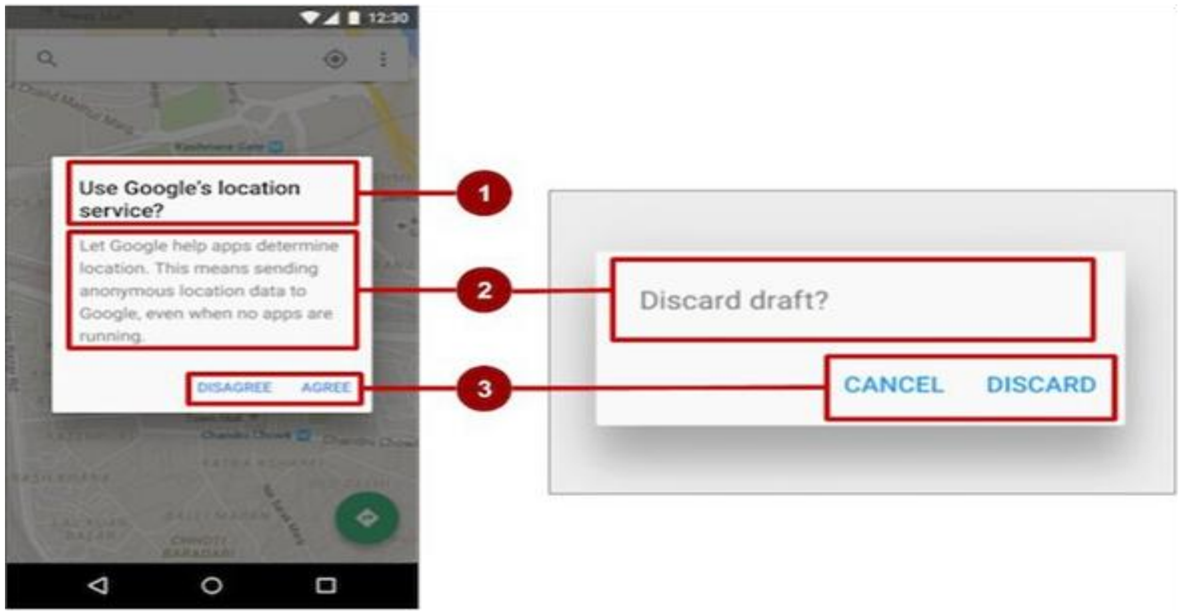


The Dialog class is the base class for dialogs, but you should avoid instantiating Dialog directly unless you are creating a custom dialog. For standard Android dialogs, use one of the following subclasses:

- AlertDialog: A dialog that can show a title, up to three buttons, a list of selectable items, or a custom layout.
- 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 inform the user about a situation as it occurs, or an action *before* it occurs (as in discarding a draft).
- You can provide buttons in an alert to make a decision. For example, an alert dialog might require the user to click **Continue** after reading it, or give the user a choice to agree with an action by clicking a positive button (such as **OK** or **Accept**), or to disagree by clicking a negative button (such as **Disagree** or **Cancel**).


Use Google's location service?

Let Google help apps determine location. This means sending anonymous location data to Google, even when no apps are running.

Discard draft?

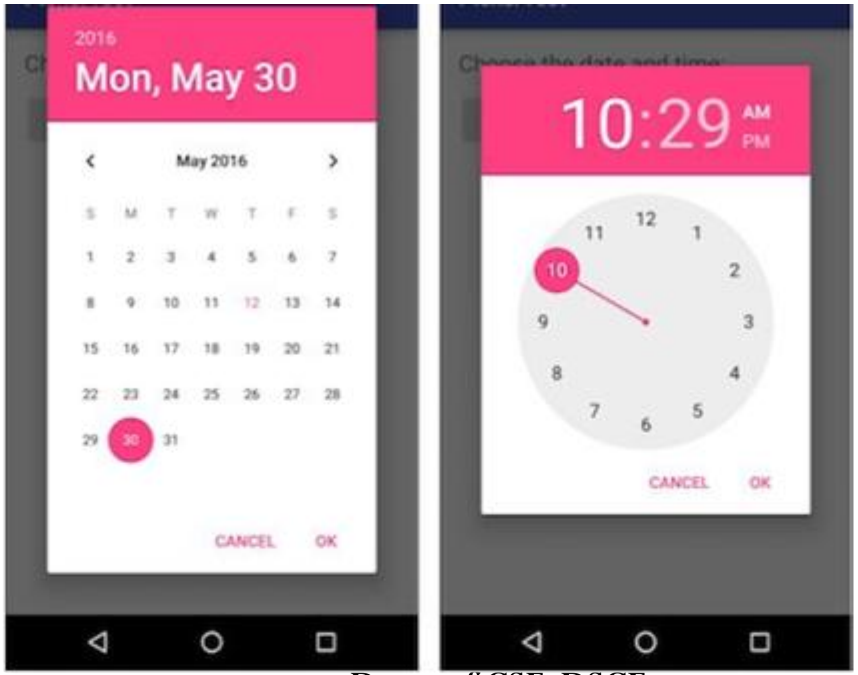
1. Title: A title is optional. Most alerts don't need titles. If you can summarize a decision in a sentence or two by either asking a question (such as, "Discard draft?") or making a statement related to the action buttons (such as, "Click OK to continue"), don't bother with a title. Use a title if the situation is high-risk, such as the potential loss of connectivity or data, and the content area is occupied by a detailed message, a list, or custom layout.

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.

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).



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:

1. Expand app > java > com.example.android.DateTimePickers and select MainActivity.

2. 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

Recognizing gestures

- A touch gesture occurs when a user places one or more fingers on the touch screen, and your app interprets that pattern of touches as a particular gesture, such as a long touch, double-tap, fling, or scroll.
- Android provides a variety of classes and methods to help you create and detect gestures. Although your app should not depend on touch gestures for basic behaviors (since the gestures may not be available to all users in all contexts), adding touch-based interaction to your app can greatly increase its usefulness and appeal.
- To provide users with a consistent, intuitive experience, your app should follow the accepted Android conventions for touch gestures.
- The Gestures design guide shows you how to design common gestures in Android apps. For more code samples and details, see Using Touch Gestures in the Android developer documentation.

Detecting all gestures

To detect all types of gestures, you need to perform two essential steps:

1. Gather data about touch events.

2. Interpret the data to see if it meets the criteria for any of the gestures your app supports.

The gesture starts when the user first touches the screen,continues as the system tracks the position of the user's finger(s), and ends by capturing the final event of the user's fingers leaving the screen. Throughout this interaction, an object of the MotionEvent class is delivered to onTouchEvent(), providing the details of every interaction.

Your app can use the data provided by the MotionEvent to determine if a gesture it cares about happened.

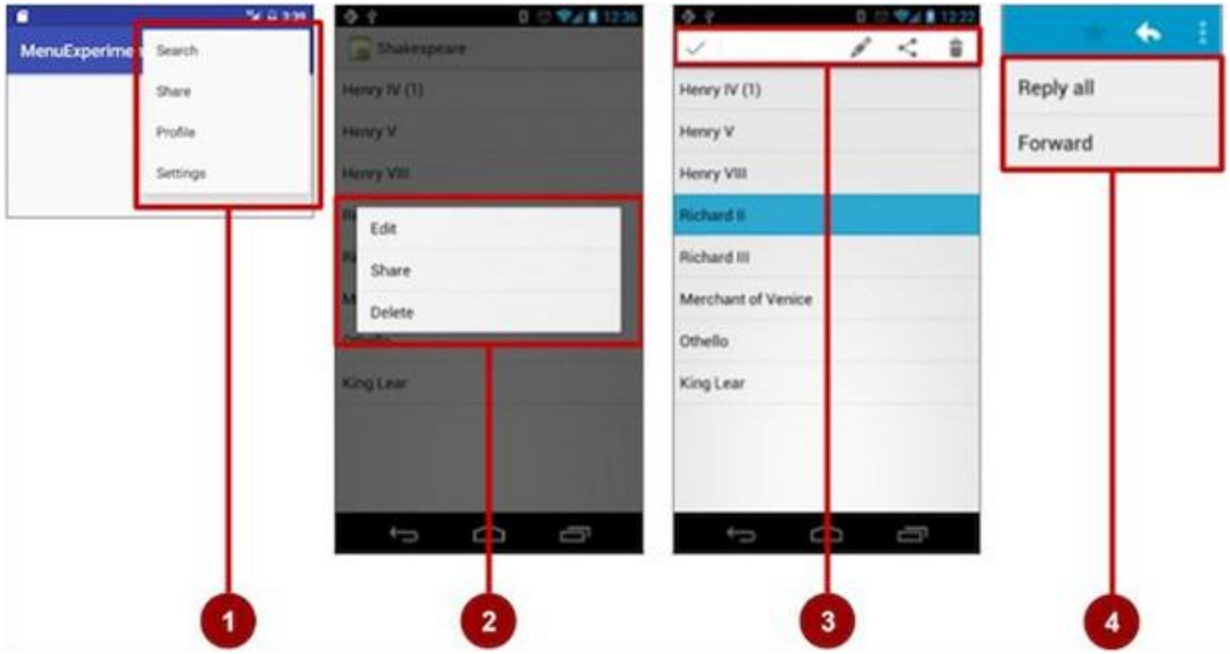
Menus

Contents:

- Types of menus
- The app bar and options menu
- Contextual menu
- Popup menu

Types of menus

- 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



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.

2. Context menu: Appears as a floating list of choices when the user performs a long tap on an element on the screen.

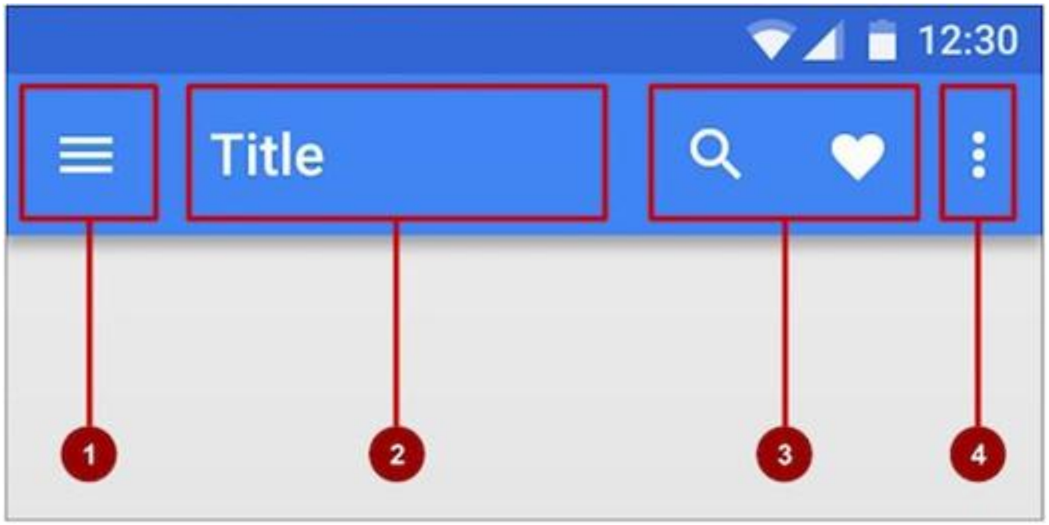
Examples of menu options: Edit to edit the element, Delete to delete it, and Share to share it over social media.

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 Image Button, 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.

The app bar and options menu

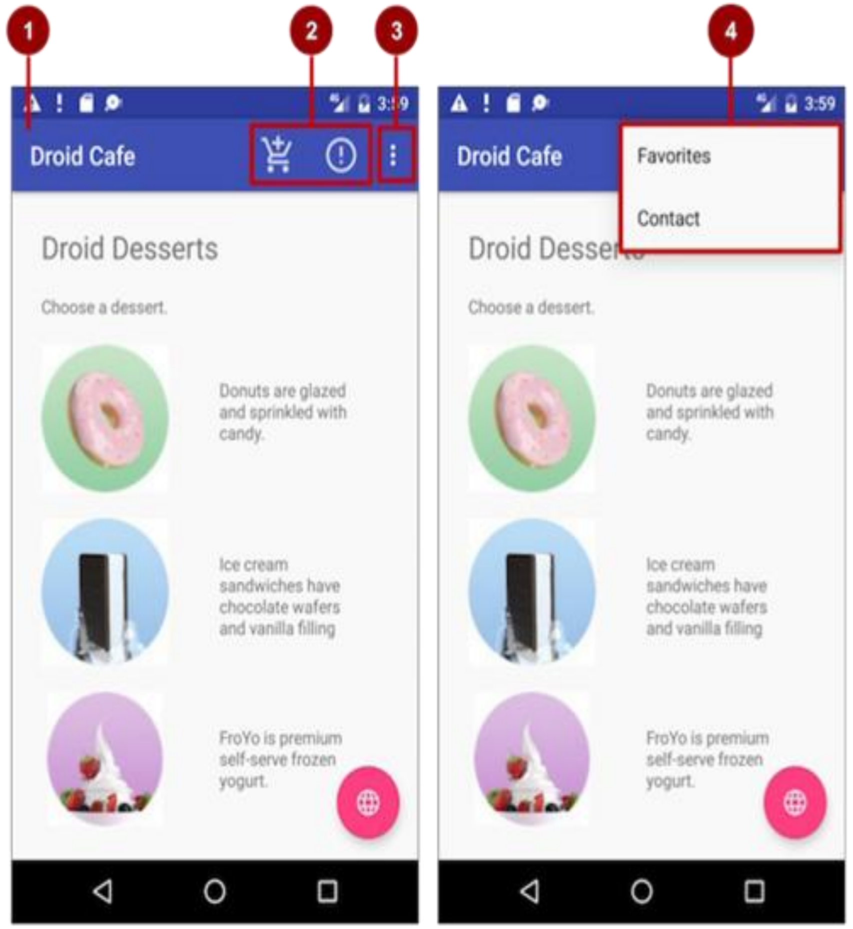
- The app bar (also called the action bar) is a dedicated space at the top of each activity screen. When you create an activity from a template (such as Empty Template), an app bar is automatically included for the activity in a Coordinator Layout root view group at the top of the view hierarchy.
- The app bar by default shows the app title, or the name defined in AndroidManifest.xml by the android:label attribute for the activity. It may also include the Up button for navigating up to the parent activity, which is described in the next chapter.
- The options menu in the app bar provides navigation to other activities in the app, or the primary options that affect using the app itself — but not ones that perform an action on an element on the screen.
- For example, your options menu might provide the user choices for navigating to other activities, such as placing an order, or for actions that have a global impact on the app, such as changing settings or profile information.



1. Navigation button or Up button: Use a navigation button in this space to open a navigation drawer, or use an Up button for navigating up through your app's screen hierarchy to the parent activity. Both are described in the next chapter.
2. Title: The title in the app bar is the app title, or the name defined in AndroidManifest.xml by the android:label attribute for the activity.
3. Action icons for the options menu: Each action icon appears in the app bar and represents one of the options menu's most frequently used items. Less frequently used options menu items appear in the overflow options menu.
4. Overflow options menu: The overflow icon opens a popup with option menu items that are not shown as icons in the app bar.

In the above figure:

1. App bar. The app bar includes the app title, the options menu, and the overflow button.
2. Options menu action icons. The first two options menu items appear as icons in the app bar.
3. Overflow button. The overflow button (three vertical dots) opens a menu that shows more options menu items.
4. Options overflow menu. After clicking the overflow button, more options menu items appear in the overflow menu.

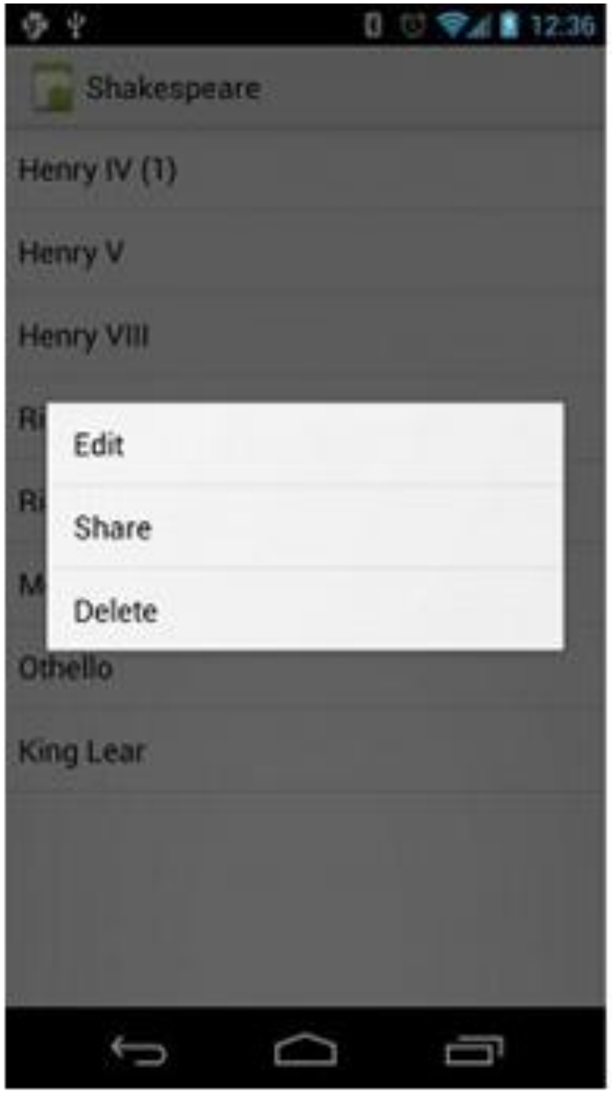

1
2
3
4
Droid Cafe
Favorites
Contact
Droid Desserts
Choose a dessert.
Donuts are glazed and sprinkled with candy.
Ice cream sandwiches have chocolate wafers and vanilla filling
FroYo is premium self-serve frozen yogurt.

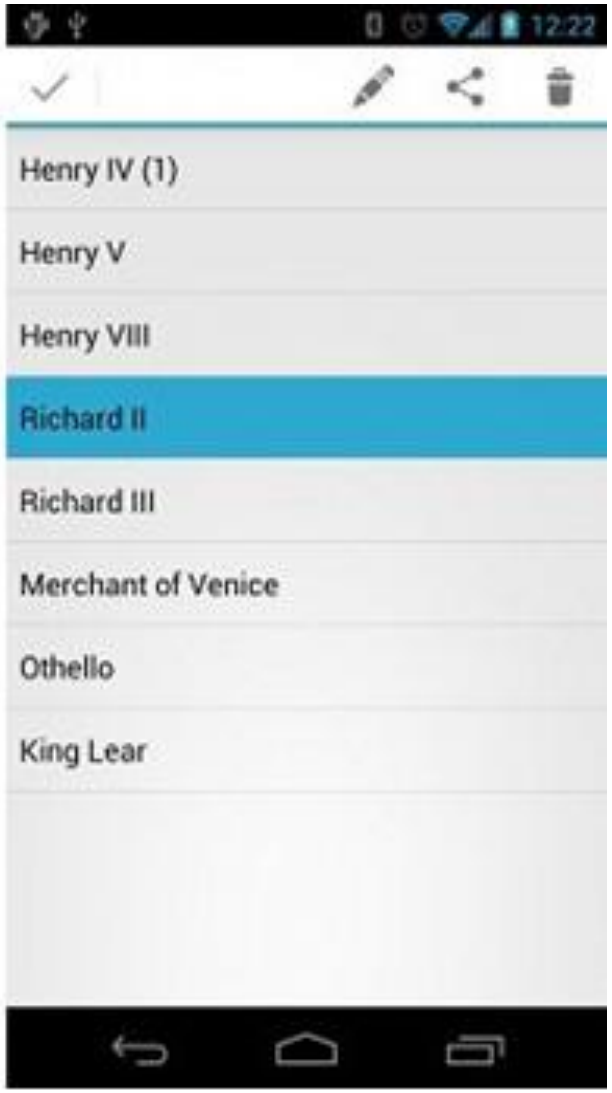
Contextual Menu

Android provides two kinds of contextual menus:

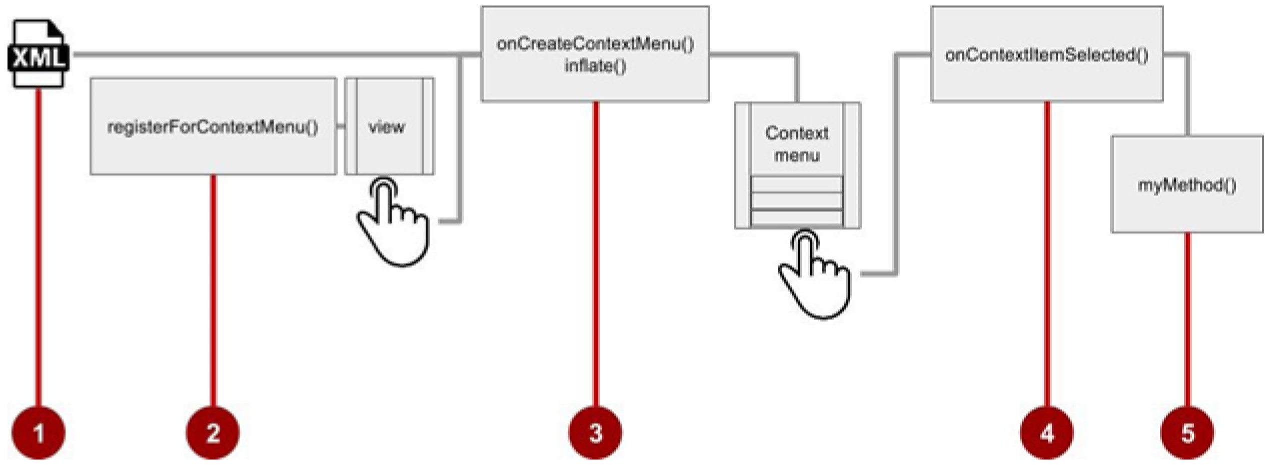
1.A context menu : Shown on the left side in the figure below, appears as a floating list of menu items when the user performs a long tap on a view element on the screen. It is typically used to modify the view element or use it in some fashion. For example, a context menu might include Edit to edit a view element, Delete to delete it, and Share to share it over social media. Users can perform a contextual action on one view element at a time.

2.A Contextual action bar : Shown on the right side of the figure below, appears at the top of the screen in place of the app bar or underneath the app bar, with action items that affect the selected view element(s). Users can perform an action on multiple view elements at once (if your app allows it).


12:36
Shakespeare
Henry IV (1)
Henry V
Henry VIII
Ri
Ri
M
Edit
Share
Delete
Othello
King Lear


12:22
Henry IV (1)
Henry V
Henry VIII
Richard II
Richard III
Merchant of Venice
Othello
King Lear

Floating context menu



Follow these steps to create a floating context menu for one or more view elements (refer to figure above):

1. Create an XML menu resource file for the menu items, and assign appearance and position attributes (as described in the previous section).

2. Register a view to the context menu using the registerForContextMenu() method of the Activity class.

3. Implement the onCreateContextMenu() method in your activity or fragment to inflate the menu.

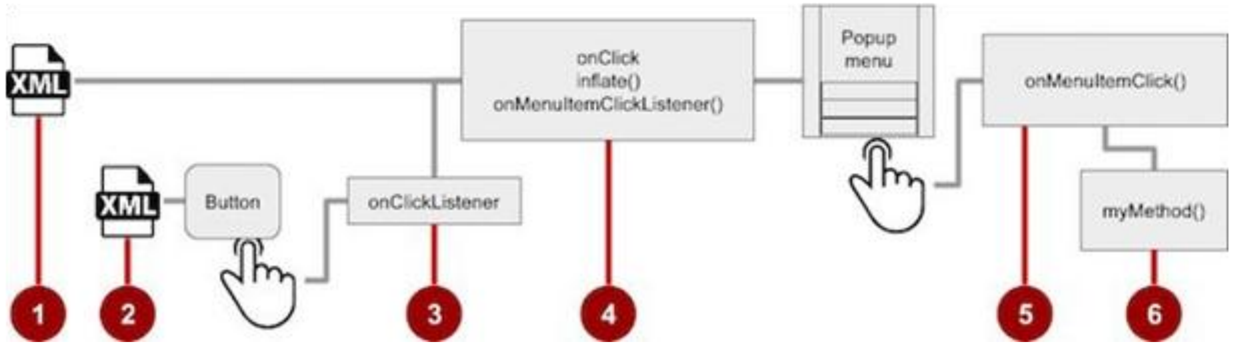
4. Implement the onContextItemSelected() method in your activity or fragment to handle menu item clicks.

5. Create a method to perform an action for each context menu item.

Popup menu

- A PopupMenu is a vertical list of items anchored to a View. It appears below the anchor view if there is room, or above the view otherwise.
- A popup menu is typically used to provide an overflow of actions (similar to the overflow action icon for the options menu) or the second part of a two-part command.
- Use a popup menu for extended actions that relate to regions of content in your activity.
- Unlike a context menu, a popup menu is anchored to a Button (View), is always available, and it's actions generally do not affect the content of the View.

Creating a pop-up menu :



Steps for Creating Pop up menu :
1. Create an XML menu resource file for the popup menu items, and assign appearance and position attributes (as
described in a previous section).
2. Add an ImageButton for the popup menu icon in the XML activity layout file.
3. Assign onClickListener() to the button.
4. Override the onClick() method to inflate the popup menu and register it with PopupMenu.OnMenuItemClickListener.
5. Implement the onMenuItemClick() method.
6. Create a method to perform an action for each popup menu item.

Screen Navigation

Following types of navigation:

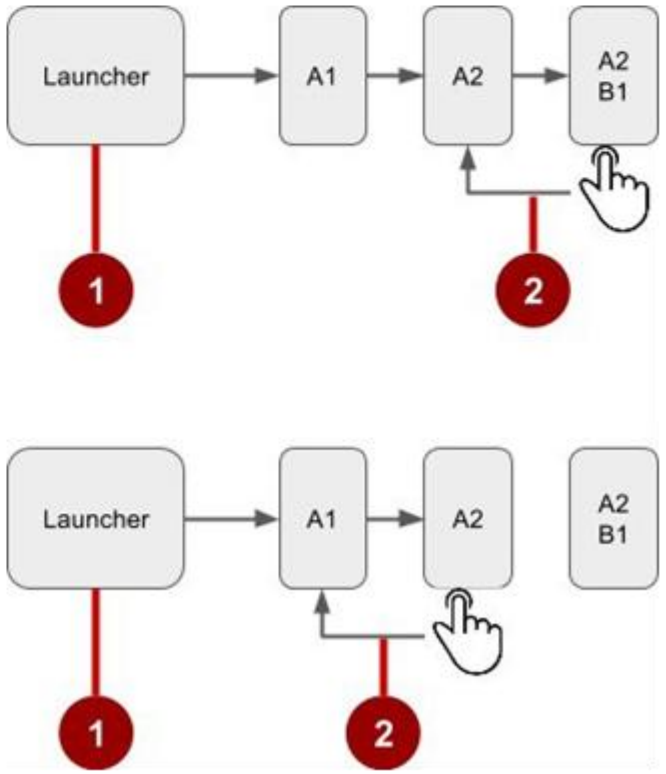
- Back navigation: Users can navigate back to the previous screen using the Back button.
- Hierarchical navigation: Users can navigate through a hierarchy of screens organized with a parent screen for every set of child screens.

Back-button navigation

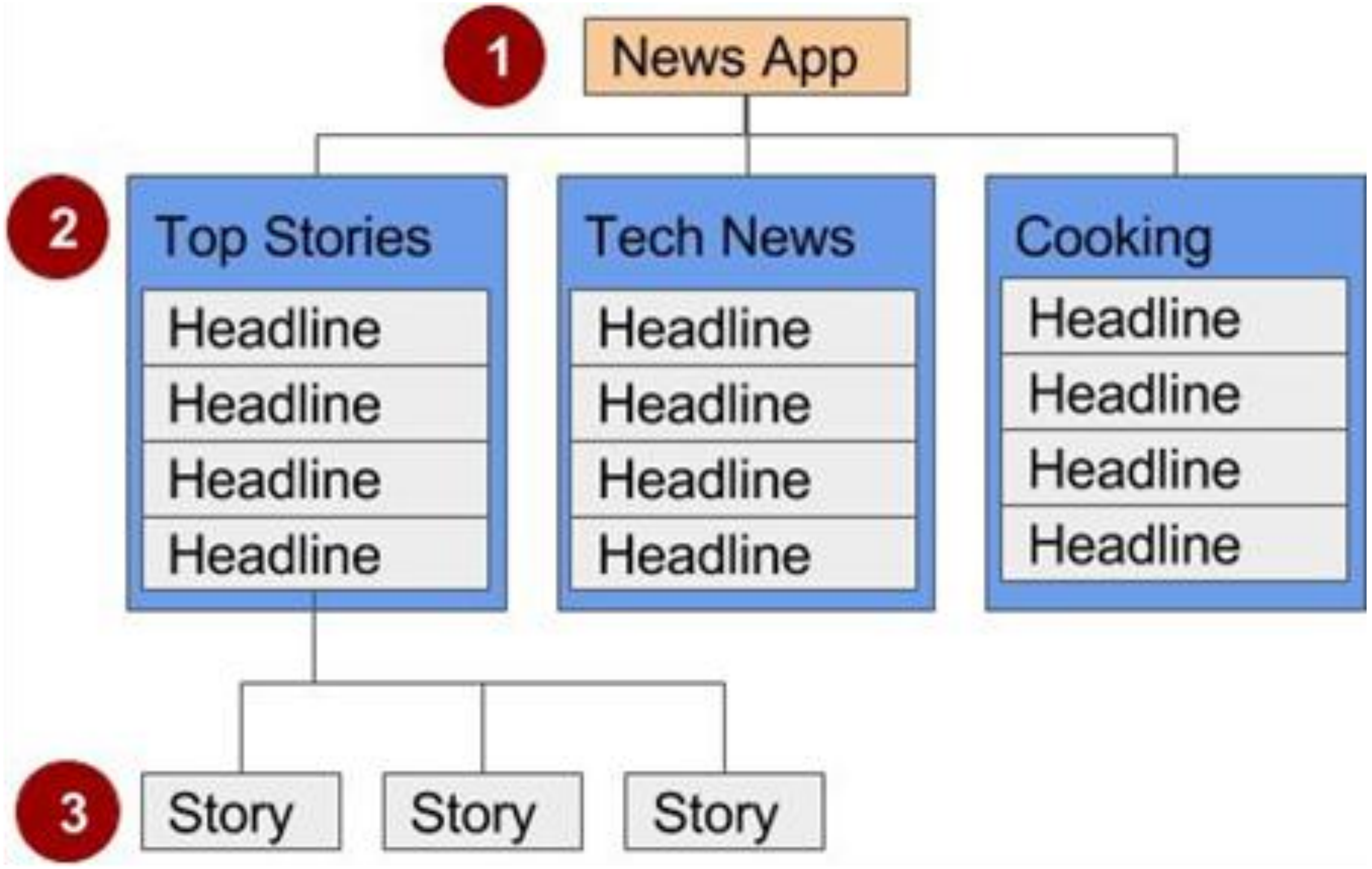
In the above figure:

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

- 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.



Hierarchical navigation patterns



In the figure above:

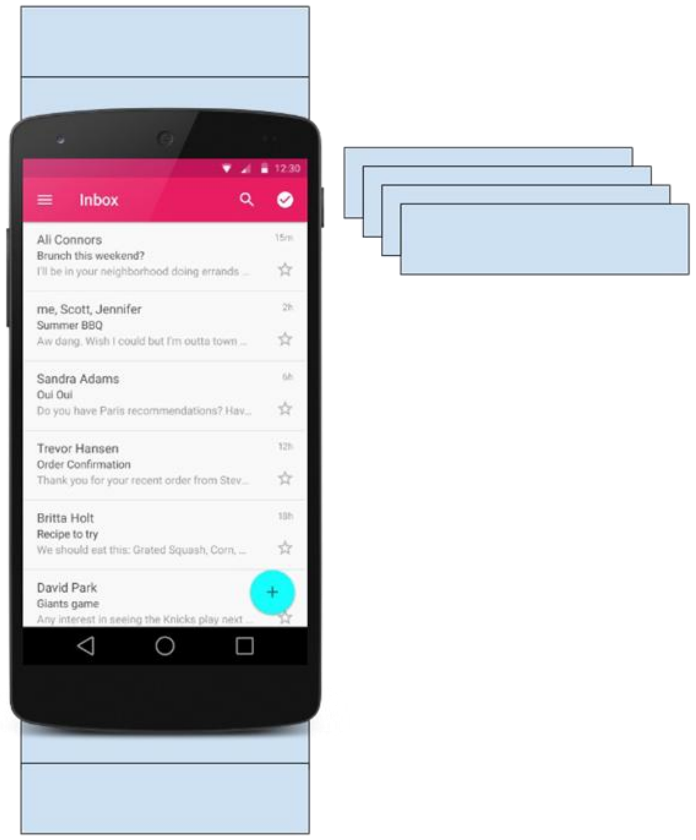
1. Parent screen. A parent screen (such as a news app's home screen) enables navigation down to child screens.main activity of an app is usually the parent screen. Implement a parent screen as an Activity with descendant navigation to one or more child screens.

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.

Recycler View

- 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 you need to display a large amount of scrollable data, or data collections whose elements change at runtime based on user action or network events.



Ali Connors 15m
Brunch this weekend?
I'll be in your neighborhood doing errands ...

me, Scott, Jennifer 2h
Summer BBQ
Aw dang. Wish I could but I'm outta town ...

Sandra Adams 6h
Oui Oui
Do you have Paris recommendations? Hav...

Trevor Hansen 12h
Order Confirmation
Thank you for your recent order from Stev...

Britta Holt 18h
Recipe to try
We should eat this: Grated Squash, Corn, ...

David Park
Giants game
Any interest in seeing the Knicks play next

Recycler View components :

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

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.

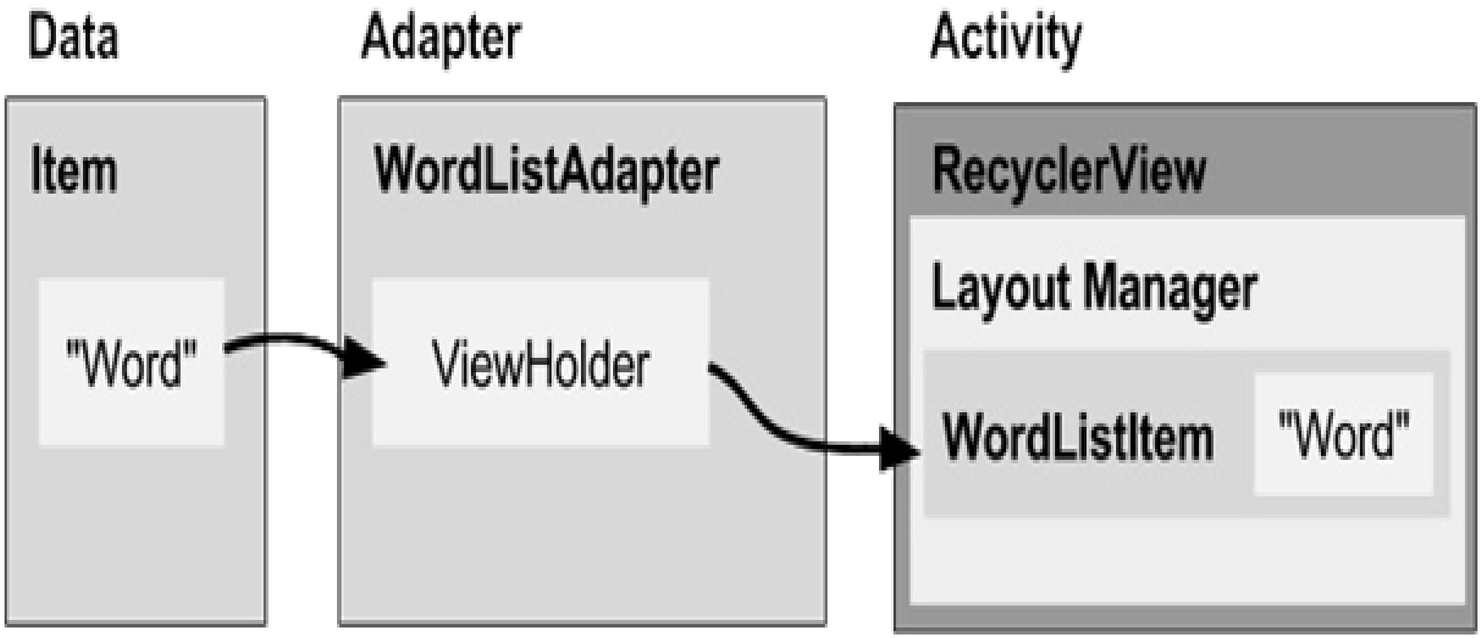
3.Layout for one item of data. All list items look the same, so you can use the same layout for all of them. The item layout has to be created separately from the activity's layout, so that one item view at a time can be created and filled with data.

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.

The diagram below shows the relationship between these components.



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.

Drawables, Styles, and Themes

Drawables

A drawable is a graphic that can be drawn to the screen. You retrieve drawables using APIs such as getDrawable(int) , and you apply a drawable to an XML resource using attributes such as android:drawable

Using drawables

To display a drawable, use the the drawable is displayed and where the drawable file is located. For example, this ImageView displays an image called "birthdaycake.png":

```
<ImageView
android:id="@+id/tiles"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:src="@drawable/birthdaycake" />
```

About the <ImageView> attributes:

1.**The android** : id attribute sets a shortcut name that you use to call the image later.

2. **android:layout_width and android:layout_height** : attributes specify the size of the View. In this example the height and width are set to wrap_content , which means the View is only big enough to enclose the image within it,plus padding.

3. **android:src** : attribute gives the location where this image is stored. If you have versions of the image that are appropriate for different screen resolutions, store them in folders named res/drawable-[density]/. For example, store a version of birthdaycake.png appropriate for hdpi screens in res/drawable hdpi/birthdaycake.png.

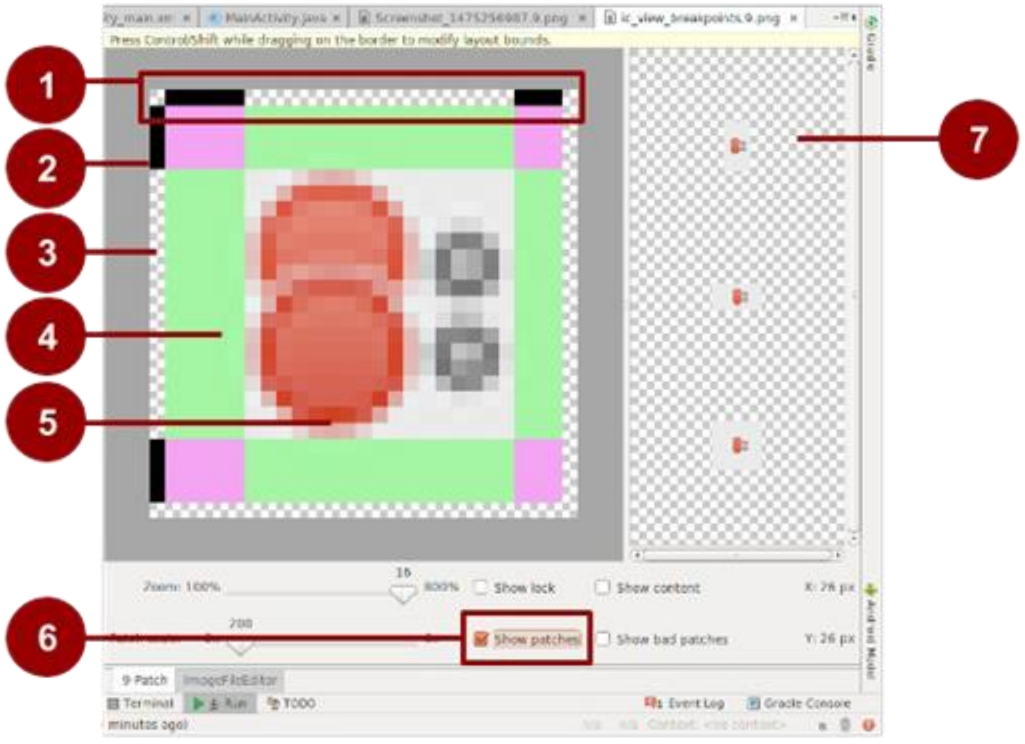
4. **<ImageView> :** also has attributes that you can use to crop your image if it is too large or has a different aspect ratio than the layout or the View. For complete details, see the ImageView class documentation.

Image files

- An image file is a generic bitmap file. Android supports image files in several formats: WebP (preferred), PNG (preferred), and JPG (acceptable). GIF and BMP formats are supported, but discouraged.
- The WebP format is fully supported from Android 4.2. WebP compresses better than other formats for lossless and lossy compression, potentially resulting in images more than 25% smaller than JPEG formats.
- You can convert existing PNG and JPEG images into WebP format before upload. For more about WebP, see the WebP documentation.

Nine-patch files

- A 9-patch is a PNG image in which you define stretchable regions. Use a 9-patch as the background image for a View to make sure the View looks correct for different screen sizes and orientations.
- For example, in a View that has layout_width set to "wrap_content" , the View stays big enough to enclose its content (plus padding).
- If you use a normal PNG image as the background image for the View, the image might be too small for the for the View on some devices, because the View stretches to accommodate the content inside it.
- If you use a 9-patch image instead, the 9-patch stretches as the View stretches.


1
2
3
4
5
6
7
Press Control/Shift while dragging on the border to modify layout bounds.
Zoom: 100%
800%
Show lock
Show content
Show patches
Show bad patches
X: 26 px
Y: 26 px
9-Patch
ImageFileEditor
Terminal
Run
TODO
Event Log
Gradle Console

1. Border to indicate which regions are okay to stretch for width (horizontally). For example, in a View that is wider than the image, the green stripes on the left- and right-hand sides of this 9-patch can be stretched to fill the View. Places that can stretch are marked with black. Click to turn pixels black.

2. Border to indicate regions that are okay to stretch for height (vertically). For example, in a View that is taller then the image, the green stripes on the top and bottom of this 9-patch can be stretched to fill the View.

3. Turn off pixels by shift-clicking (ctrl-click on Mac).

4. Stretchable area.

5. Not stretchable.

6. Check Show patches to preview the stretchable patches in the drawing area.

7. Previews of stretched image. Tip: Make sure that stretchable regions are at least 2x2 pixels in size. Otherwise, they may disappear when the image is scaled down.

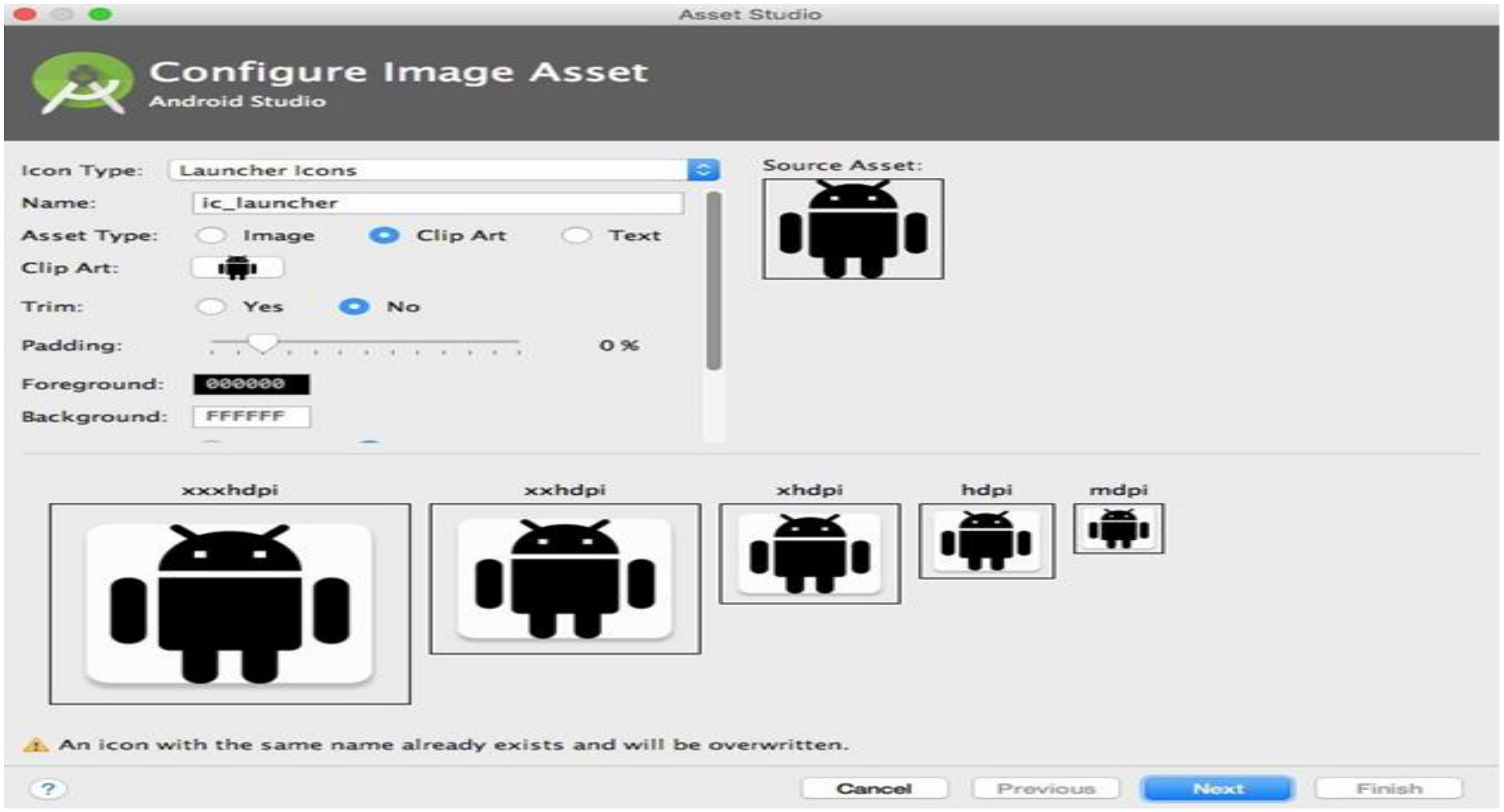
Image Asset Studio

- Android Studio includes a tool called Image Asset Studio that helps you generate your own app icons from Material Design icons, custom images, and text strings.
- It generates a set of icons at the appropriate resolution for each generalized screen density that your app supports.
- Image Asset Studio places the newly generated icons in density-specific folders under the res/ folder in your project.
- At runtime, Android uses the appropriate resource based on the screen density of the device your app is running on.

Image Asset Studio helps you generate the following icon types:

- Launcher icons
- Action bar and tab icons
- Notification icons

To use Image Asset Studio, right-click on the res/ folder in Android Studio and select New > Image Asset. The Configure Asset Studio wizard opens and guides you through the process.


Asset Studio

Configure Image Asset
Android Studio

Icon Type: Launcher Icons
Name: ic_launcher
Asset Type: Image [✓ Clip Art] Text
Clip Art:
Trim: Yes [✓ No]
Padding: 0 %
Foreground: 000000
Background: FFFFFF

An icon with the same name already exists and will be overwritten.

Styles

- In Android, a style is a collection of attributes that define the look and format of a View.
- You can apply the same style to any number of Views in your app; for example, several TextViews might have the same text size and layout.
- Using styles allows you to keep these common attributes in one location and apply them to each TextView using a single line of code in XML.

Defining and applying styles :

To create a style, add a <style> element inside a <resources> element in any XML file located in the res/values/ folder.

When you create a project in Android Studio, a res/values/styles.xml file is created for you.

A <style> element includes the following:

- A name attribute. Use the style's name when you apply the style to a View.
- An optional parent attribute. You learn about using parent attributes in the Inheritance section below.
- Any number of <item> elements as child elements of <style> . Each <item> element includes one style attribute.

Inheritance :

- A new style can inherit the properties of an existing style. When you create a style that inherits properties, you define only the properties that you want to change or add.
- You can inherit properties from platform styles and from styles that you create yourself. To inherit a platform style, use the parent attribute to specify the resource ID of the style you want to inherit.
- For example, here's how to inherit the Android platform's default text appearance (the TextAppearance style) and change its color:

```
<style name="GreenText"
parent="@android:style/TextAppearance">
<item name="android:textColor">#00FF00</item>
</style>
```

To apply this style, use @style/GreenText . To inherit a style that you created yourself, use the name of the style you want to inherit as the first part of the new style's name, and separate the parts with a period:

name="StyleToInherit.Qualifier"

Themes

You create a theme the same way you create a style, which is by adding a <style> element inside a <resources> element in any XML file located in the res/values/ folder.

What's the difference between a style and a theme?

- A style applies to a View. In XML, you apply a style using the style attribute.
- A theme applies to an entire Activity or application, rather than to an individual View. In XML, you apply a theme using the android:theme attribute.

Any style can be used as a theme. For example, you could apply the CodeFont style as a theme for an Activity, and all the text inside the Activity would use gray monospace font.

Applying Themes :

To apply a theme to your app, declare it inside an <application> element inside the AndroidManifest.xml file. This example applies the AppTheme theme to the entire application:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.exampledomain.myapp">
<application
...
android:theme="@style/AppTheme">
</application>
...
```

To apply a theme to an Activity, declare it inside an <activity> element in the AndroidManifest.xml file.

In this example,

the android:theme attribute applies the Theme_Dialog platform theme to the Activity:

```
<activity android:theme="@android:style/Theme.Dialog">
```

Default theme

When you create a new project in Android Studio, a default theme is defined for you within the styles.xml file. For example, this code might be in your styles.xml file:

```
<style name="AppTheme"
parent="Theme.AppCompat.Light.DarkActionBar">
<!-- Customize your theme here. -->
<item name="colorPrimary">@color/colorPrimary</item>
<item name="colorPrimaryDark">@color/colorPrimaryDark</item>
<item name="colorAccent">@color/colorAccent</item>
</style>
```

In this example, AppTheme inherits from Theme.AppCompat.Light.DarkActionBar , which is one of the many Android platform themes available to you. (You'll learn about the color attributes in the unit on Material Design.)

Material Design

- Material Design is a visual design philosophy that Google created in 2014.
- The aim of Material Design is a unified user experience across platforms and device sizes.
- Material Design includes a set of guidelines for style, layout, motion, and other aspects of app design. The complete guidelines are available in the Material Design Spec.
- Material Design is for desktop web applications as well as for mobile apps .

Principles of Material Design

- **The "material" metaphor :**

In Material Design, elements in your Android app behave like real world materials: they cast shadows, occupy space, and interact with each other.

- **Bold, graphic, intentional :**

Material Design involves deliberate color choices, edge-to-edge imagery, large-scale typography, and intentional white space that create a bold and graphic interface.

- **Meaningful motion**

Make animations and other motions in your app meaningful, so they don't happen at random. Use motions to reinforce the idea that the user is the app's primary mover. For example, design your app so that most motions are initiated by the user's actions, not by events outside the user's control. You can also use motion to focus the user's attention, give the user subtle feedback, or highlight an element of your app.

Colors

Material Design color palette

Material Design principles include the use of bold color. The Material Design color palette contains colors to choose from, each with a primary color and shades labeled from 50 to 900:

- Choose a color labeled "500" as the primary color for your brand. Use that color and shades of that color in your app.
- Choose a contrasting color as your accent color and use it to create highlights in your app. Select any color that starts with "A."

When you create an Android project in Android Studio, a sample Material Design color scheme is selected for you and applied to your theme. In values/colors.xml, three <color> elements are defined, colorPrimary , colorPrimaryDark , and colorAccent

In values/styles.xml, the three defined colors are applied to the default theme, which applies the colors to some app elements by default:

1.**colorPrimary** is used by several Views by default. For example, in the AppTheme theme, colorPrimary is used as the background color for the action bar. Change this value to the "500" color that you select as your brand's primary color.

2. **colorPrimaryDark** is used in areas that need to slightly contrast with your primary color, for example the status bar. Set this value to a slightly darker version of your primary color.

3. **colorAccent** is used as the highlight color for several Views. It's also used for switches in the "on" position, floating action buttons, and more.

Contrast

Make sure all the text in your app's UI contrasts with its background. Where you have a dark background, make the text on top of it a light color, and vice versa. This kind of contrast is important for readability and accessibility, because not all people see colors the same way. If you use a platform theme such as Theme.AppCompat , contrast between text and its background is handled for you.

For example:

- If your theme inherits from Theme.AppCompat , the system assumes you are using a dark background. Therefore all of the text is near white by default.
- If your theme inherits from Theme.AppCompat.Light , the text is near black, because the theme has a light background.
- If you use the Theme.AppCompat.Light.DarkActionBar theme, the text in the action bar is near white, to contrast with the action bar's dark background. The rest of the text in the app is near black, to contrast with the light background.

Animations :

There are three ways you can create animation in your app:

1. Property animation changes an object's properties over a specified period of time.The property animation system was introduced in Android 3.0 (API level 11). Property animation is more flexible than view animation, and it offers more features.

2.View animation calculates animation using start points, end points, rotation, and other aspects of animation. The Android view animation system is older than the property animation system and can only be used for Views. It's relatively easy to set up and offers enough capabilities for many use cases.

3. Drawable animation lets you load a series of drawable resources one after another to create an animation. Drawable animation is useful if you want to animate things that are easier to represent with drawable resources, such as a progression of bitmap images.

Providing Resources for Adaptive Layouts

An adaptive layout is a layout that works well for different screen sizes and orientations, different devices, different locales and languages, and different versions of Android.

Externalizing resources

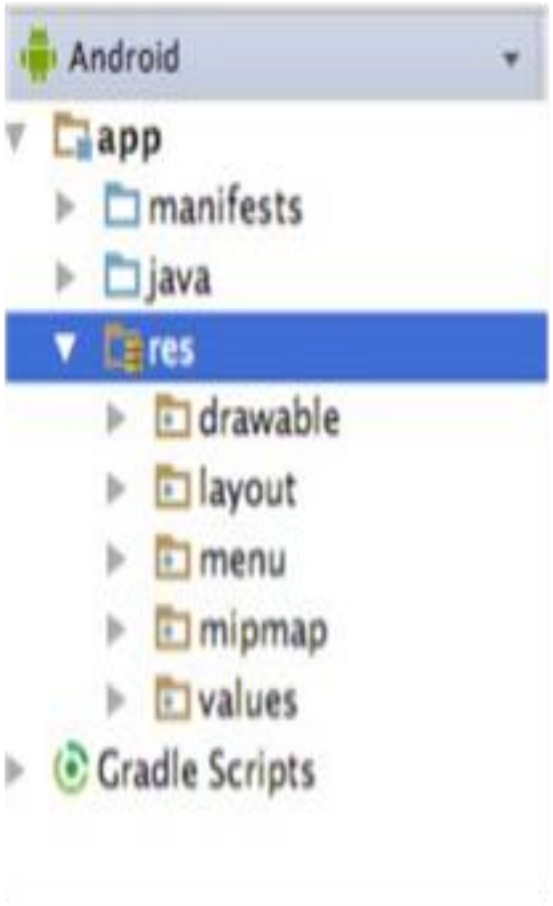
When you externalize resources, you keep them separate from your application code. For example, instead of hard-coding a string into your code, you name the string and add it to the res/values/strings.xml file. Always externalize resources such as drawables, icons, layouts, and strings. Here's why it's important:

- You can maintain externalized resources separately from your other code. If a resource is used in several places in your code and you need to change the resource, you only need to change it in one place.

- You can provide alternative resources that support specific device configurations, for example devices with different languages or screen sizes. This becomes increasingly important as more Android-powered devices become available.

Grouping resources

- Store all your resources in the res/ folder. Organize resources by type into folders under /res. You must use standardized names for these folders.
- For example, the screenshot below shows the file hierarchy for a small project, as seen in the "Android" Project view in Android Studio. The folders that contain this project's default resources use standardized names: drawable , layout



Standard Resource Folder Names

Name	Resource Type
animator/	XML files that define property animations.
anim/	XML files that define tween animations.
color/	XML files that define "state lists" of colors. (This is different from the colors.xml file in the values/ folder.) See Color State List Resource.
drawable/	Bitmap files (WebP, PNG, 9-patch, JPG, GIF) and XML files that are compiled into drawables. See Drawable Resources.
mipmap/	Drawable files for different launcher icon densities. See Projects Overview.
layout/	XML files that define user interface layouts. See Layout Resource.
menu/	XML files that define application menus. See Menu Resource.
raw/	
values/	
xml/	

Alternative resources

- Most apps provide alternative resources to support specific device configurations.
- For example, your app should include alternative drawable resources for different screen densities, and alternative string resources for different languages.
- At runtime, Android detects the current device configuration and loads the appropriate resources.
- If no resources are available for the device's specific configuration, Android uses the default resources that you include in your app—the default drawables, which are in the res/drawable/ folder, the default text strings, which are in the res/values/strings.xml file, and so on.

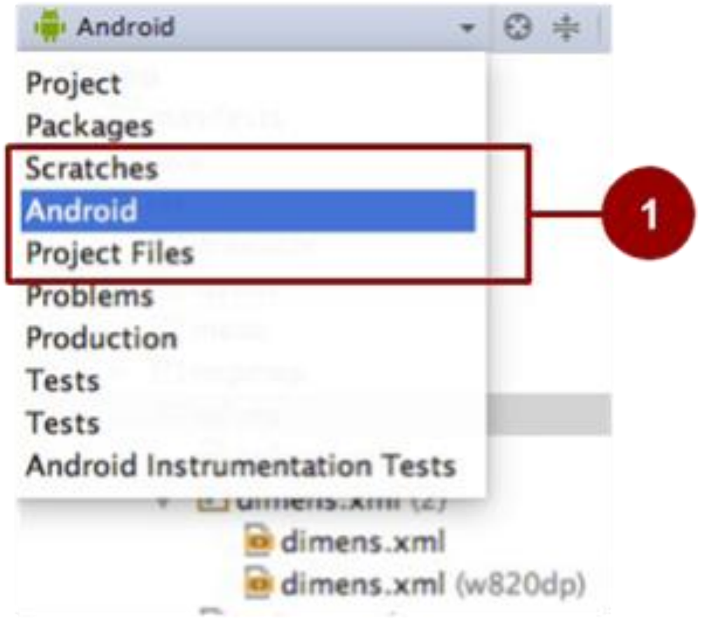
Qualifiers for Naming Alternative Resources :

Precedence	Qualifier	Description
1	MCC and MNC	The mobile country code (MCC), optionally followed by mobile network code(MNC) from the SIM card in the device. For example, mcc310 is U.S. on any carrier, mcc310-mnc004 is U.S. on Verizon, and mcc208-mnc00 is France on Orange.
2	Localization	Language, or language and region. Examples: en , en-rUS , fr-rFR , fr-rCA . Described in Localization, below
3	Layout direction	The layout direction of your application. Possible values include ldltr (layout direction left-to-right, which is the default) and ldrtl (layout direction right-to-left). To enable right-to-left layout features, set supportsRtl to "true" targetSdkVersion to 17 or higher.
4	Smallest width	Fundamental screen size as indicated by the shortest dimension of the available screen area. Example: sw320dp . Described in Smallest width, below.
5	Available width	Minimum available screen width at which the resource should be used. Specified in dp units. The format is wdp , for example, w720dp and w1024dp .
6	Available height	Minimum available screen height at which the resource should be used. Specified in dp units. The format is hdp , for example, h720dp and h1024dp

Creating alternative resources

To create alternative resource folders most easily in Android Studio, use the "Android" view in the Project tool window.

1. Selecting the "Android" view in Android Studio. If you don't see these options, make sure the Project tool window is visible by selecting View > Tool Windows > Project.



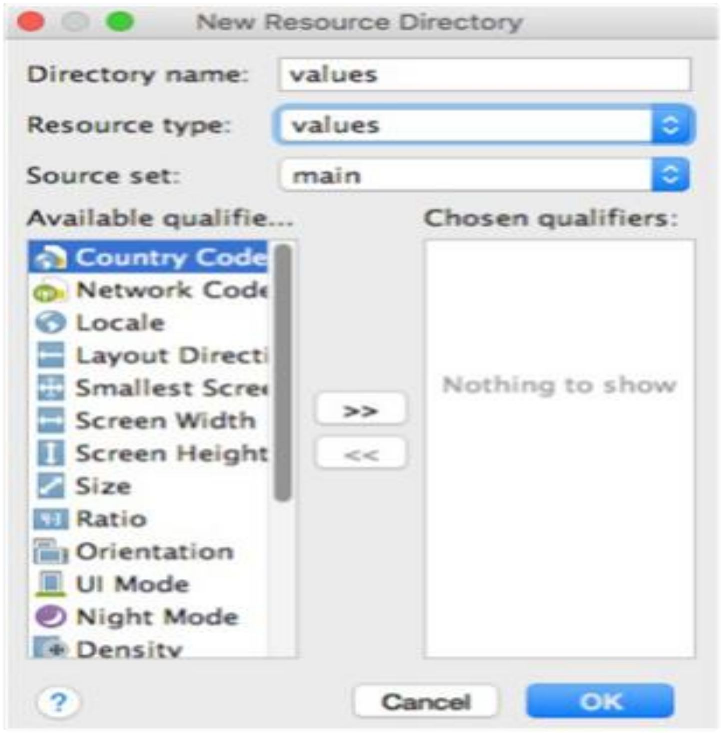
To use Android Studio to create a new configuration-specific alternative resource folder in res/:

1. Be sure you are using the "Android" view, as shown above.

2. Right-click on the res/ folder and select New > Android resource directory. The New Resource Directory dialog boxappears.

3. Select the type of resource (described in Table 1) and the qualifiers (described in Table 2) that apply to this set of alternative resources.

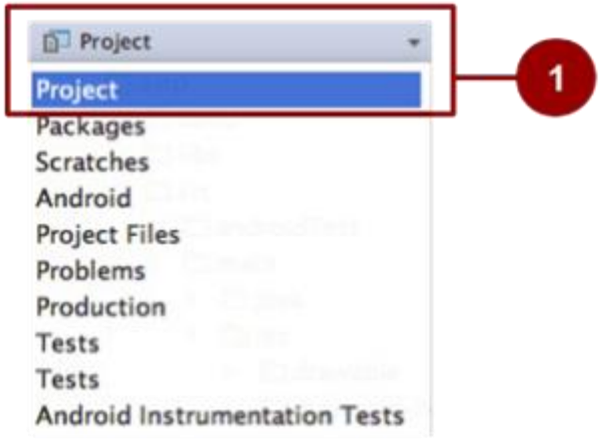
4. Click OK.


New Resource Directory

Directory name: values

Resource type: values

Source set: main



Testing the User Interface

Properly testing a user interface

- Writing and running tests are important parts of the Android app development cycle.
- Well-written tests can help you catch bugs early in your development cycle, where they are easier to fix, and helps give you confidence in your code.
- User interface (UI) testing focuses on testing aspects of the user interface and interactions with users.
- Recognizing and acting on user input is a high priority in user interface testing and validation.
- You need to make sure that your app not only recognizes the type of input but also acts accordingly. As a developer, you should get into the habit of testing user interactions to ensure that users don't encounter unexpected results or have a poor experience when interacting with your app.
- User interface testing can help you recognize the input controls where unexpected input should be handled gracefully or should trigger input validation.

Manual Testing

As the developer of an app, you probably test each UI component manually as you add the component to the app's UI. As development continues, one approach to UI testing is to simply have a human tester perform a set of user operations on the target app and verify that it is behaving correctly.

Manual testing fall into two categories:

1. Domain size: A UI has a great deal of operations that need testing. Even a relatively small app can have hundreds of possible UI operations. Over a development cycle a UI may change significantly, even though the underlying app doesn't change. Manual tests with instructions to follow a certain path through the UI may fail over time, because abutton, menu item, or dialog may have changed location or appearance.

2. Sequences: Some functionality of the app may only be accomplished with a sequence of UI events. For example, to add an image to a message about to be sent, a user may have to tap a camera button and use the camera to take a picture, or a photo button to select an existing picture, and then associate that picture with the message—usually by tapping a share or send button. Increasing the number of possible operations also increases the sequencing problem.

Automated testing

- When you automate tests of user interactions, you free yourself and your resources for other work.
- To generate a set of test cases, test designers attempt to cover all of the functionality of the system and fully exercise the UI.
- Performing all of the UI interactions automatically makes it easier to run tests for different device states (such as orientations) and different configurations.

Types of automated UI tests:

1. UI tests that work within a single app: Verifies that the app behaves as expected when a user performs a specific action or enters a specific input. It allows you to check that the app returns the correct UI output in response to user interactions in the app's activities. UI testing frameworks like Espresso allow you to programmatically simulate user actions and test complex intra-app user interactions.

2. UI tests that span multiple apps: Verifies the correct behavior of interactions between different user apps or between user apps and system apps. For example, you can test an app that launches the Maps app to show directions, or that launches the Android contact picker to select recipients for a message. UI testing frameworks that support cross-app interactions, such as UI Automator, allow you to create tests for such user-driven scenarios.

Setting up your test environment

To use the Espresso and UI Automator frameworks, you need to store the source files for instrumented tests at module name/src/androidTests/java/. This directory already exists when you create a new Android Studio project. In the Project view of Android Studio, this directory is shown in app > java as module-name (androidTest).

You also need to do the following:

- Install the Android Support Repository and the Android Testing Support Library.
- Add dependencies to the project's build.gradle file.
- Create test files in the androidTest directory.

Installing the Android Support Repository and Testing Support Library :

You may already have the Android Support Repository and its Android Testing Support Library installed with Android Studio.

To check for the Android Support Repository, follow these steps:

1. In Android Studio choose Tools > Android > SDK Manager.
2. Click the SDK Tools tab, and look for the Support Repository.

References/Bibliography

Text Book :

1. Google developer training android developer fundamentals course