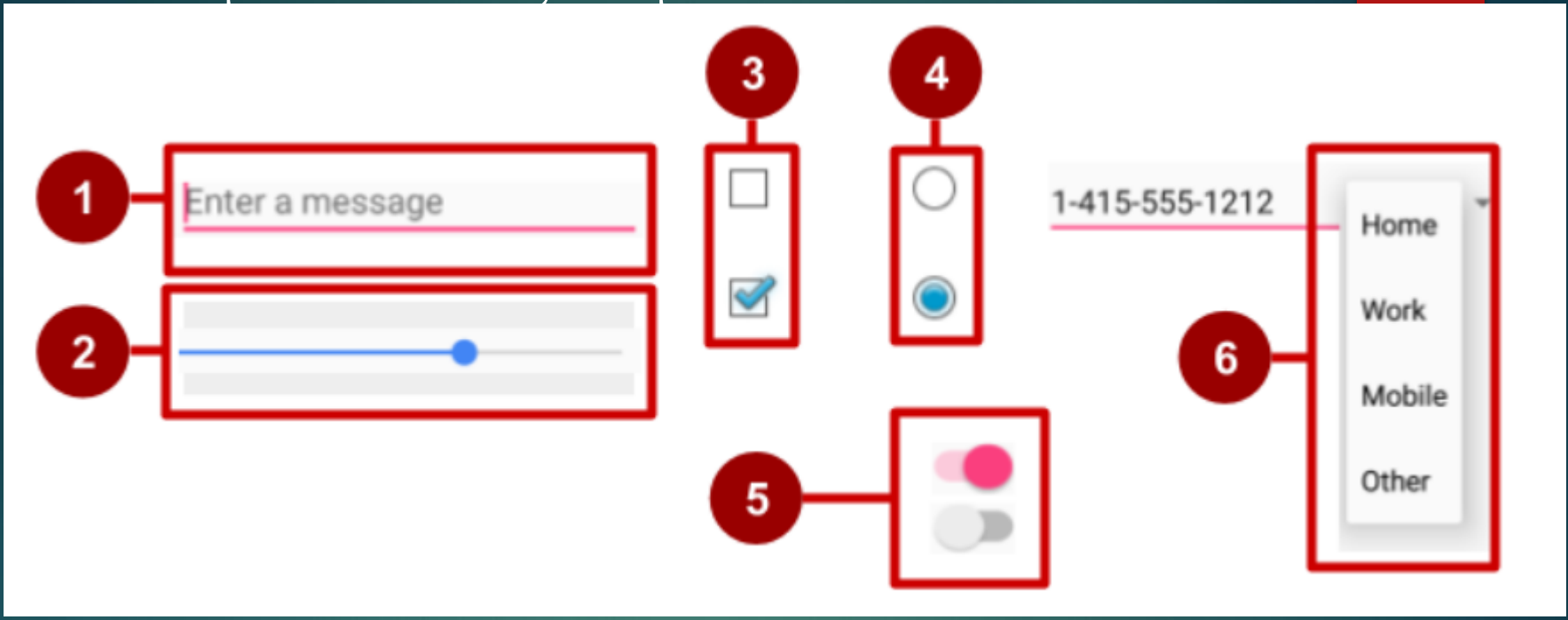
Input Controls

- Input controls are interactive elements in app's UI that accept data input
- Users input data to apps by entering text or numbers into fields using the on-screen keyboard
- Users also select options from checkboxes, radio buttons, and drop-down menus, and they change settings and turn on or turn off certain features

Android provides a variety of input controls for UI



- EditText field (subclass of TextView) for entering text using a keyboard
- SeekBar for sliding left or right to a setting
- CheckBox elements for selecting one or more options
- RadioGroup of RadioButton elements for selecting one option
- Switch for turning on or turning off an option
- Spinner drop-down menu for selecting one option

Input controls for making choices

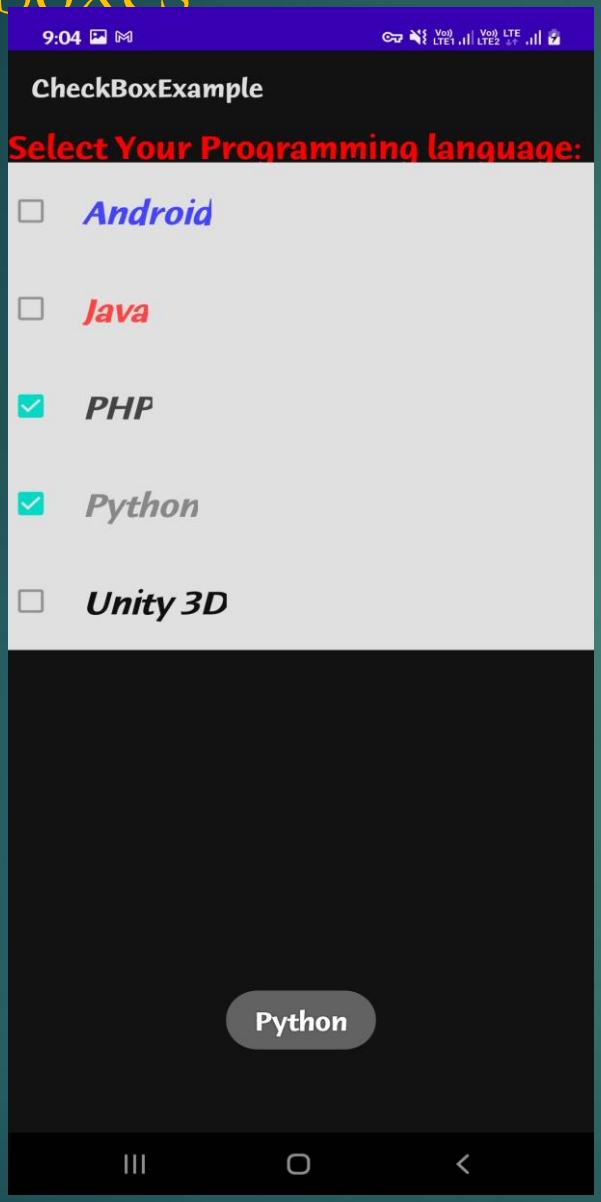
- Android offers ready-made input controls for the user to select one or more choices:

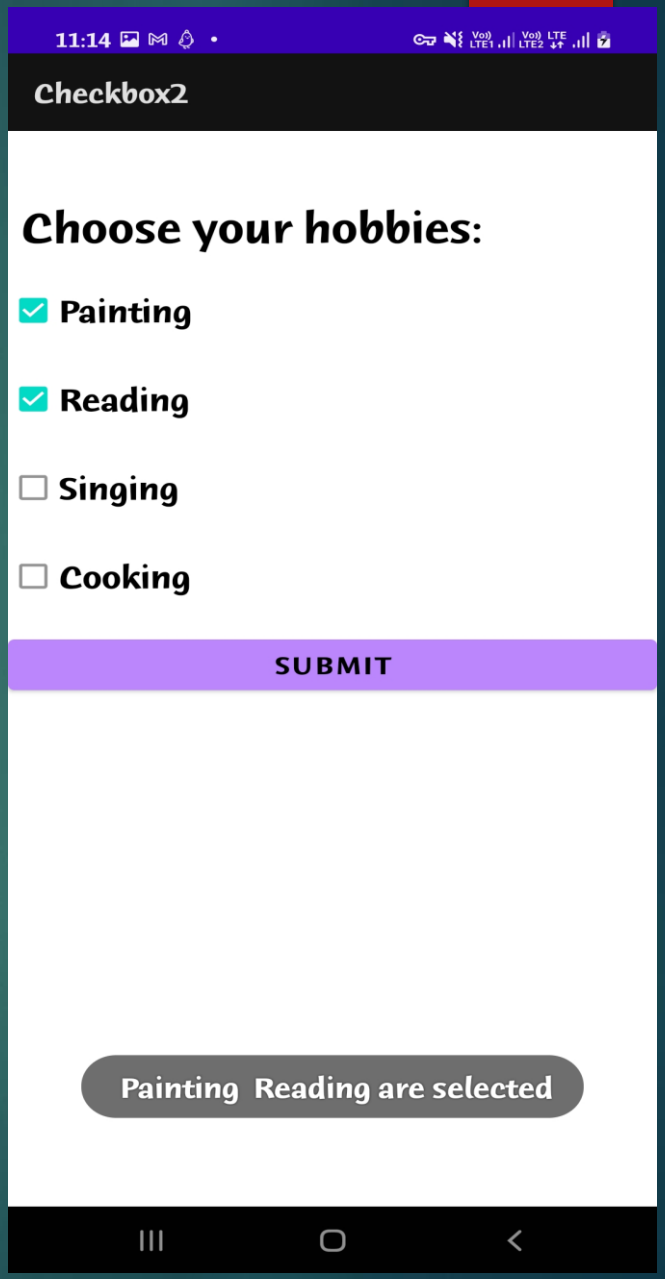
- Checkbox: Select one or more choices from a set of choices by tapping or clicking checkboxes
- RadioGroup of radio buttons: Select one choice from a set of choices by clicking one circular "radio" button. Radio buttons are useful if you are providing only two or three choices
- ToggleButton and Switch: Turn an option on or off
- Spinner: Select one choice from a set of choices in a drop-down menu. A Spinner is useful for three or more choices, and takes up little room in your layout

Checkboxes

- We use a set of checkboxes when we want the user to select any number of choices, including zero choices
- Each checkbox is independent of the other boxes in the set, so selecting one box doesn't clear the other boxes
- A user can clear a checkbox that was already selected
- Each checkbox is a separate CheckBox element in XML layout

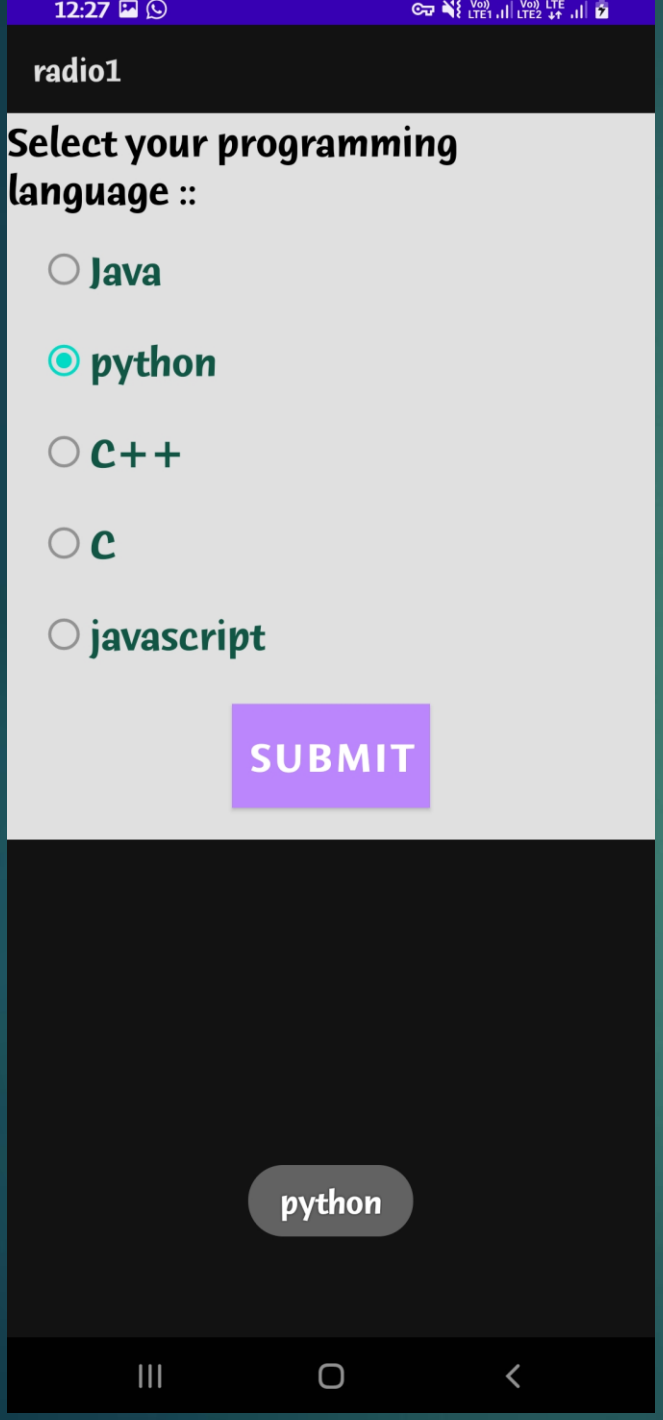
Checkboxes

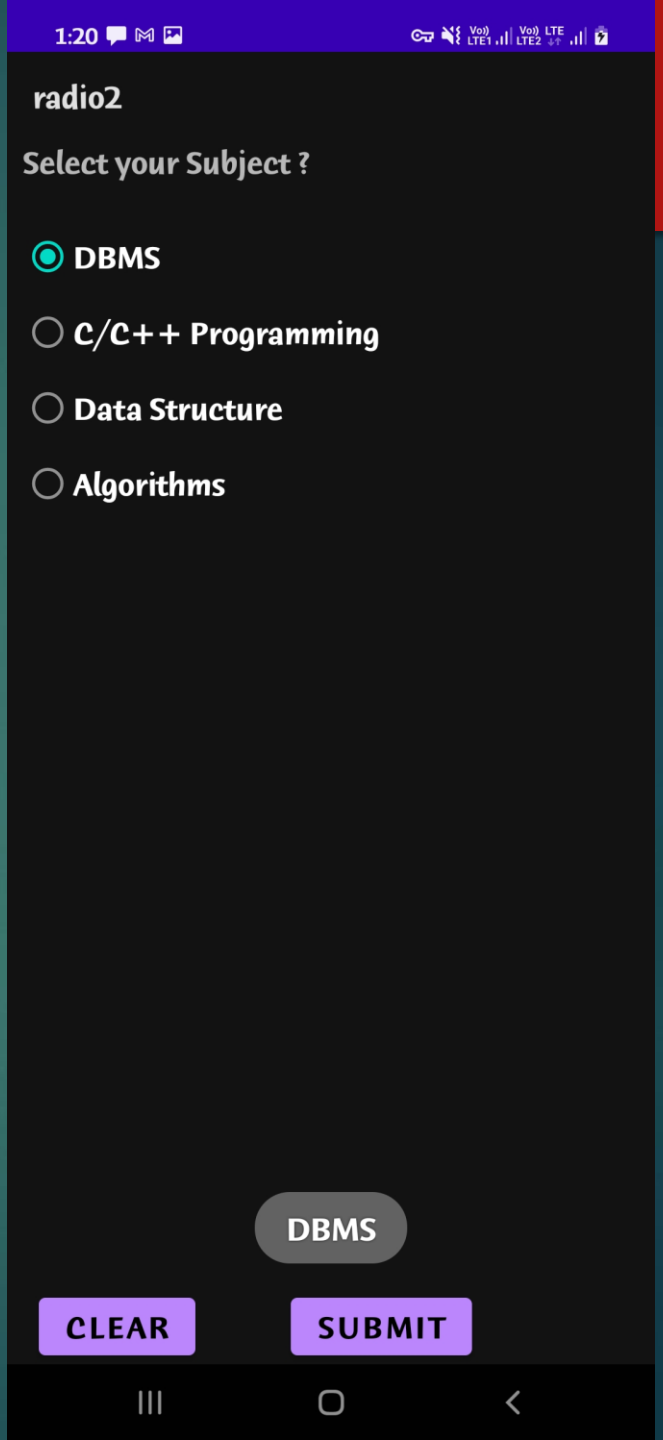

9:04
CheckBoxExample
Select Your Programming language:
Android
Java
PHP
Python
Unity 3D
Python


11:14
Checkbox2
Choose your hobbies:
Painting
Reading
Singing
Cooking
SUBMIT
Painting Reading are selected

Radio buttons

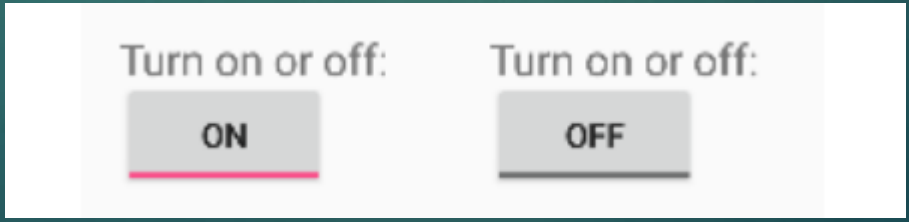
- We use radio buttons when we have two or more options that are mutually exclusive
- When the user selects one, the others are automatically deselected
- Each radio button is an instance of the RadioButton class
- Radio buttons are normally placed within a RadioGroup in a layout
- When several RadioButton elements are inside a RadioGroup, selecting one RadioButton clears all the others
- We need to add RadioButton elements to XML layout within a RadioGroup


12:27
radio1
Select your programming language ::
Java
python
C++
C
javascript
SUBMIT
python


1:20
radio2
Select your Subject ?
DBMS
C/C++ Programming
Data Structure
Algorithms
DBMS
CLEAR
SUBMIT

Toggle buttons

- A toggle input control lets the user change a setting between on and off. Android provides the ToggleButton class, which shows a raised button with "OFF" and "ON".
- It indicator light indicating the current state of ToggleButton
- Examples of toggles include the on/off switches for Wi-Fi, Bluetooth, and other options in the Settings app
- It is a subclass of compoundButton
- Programmatically, isChecked() method is used to check the current state of the toggle button. This method returns a boolean value.



Toggle buttons

checked attribute:

- checked is an attribute of toggle button used to set the current state of a toggle button
- The value should be true or false where true shows the checked state and false shows unchecked state of a toggle button. The default value of checked attribute is false
- Programmatically, isChecked() method is used to check the current state of the toggle button. This method returns a boolean value

textOn And textOff:

- textOn attribute is used to set the text when toggle button is in checked/on state. `android:textOn="Enable"`
- simpleToggleButton.setTextOn("Enable");

Toggle buttons

textOn And textOff:

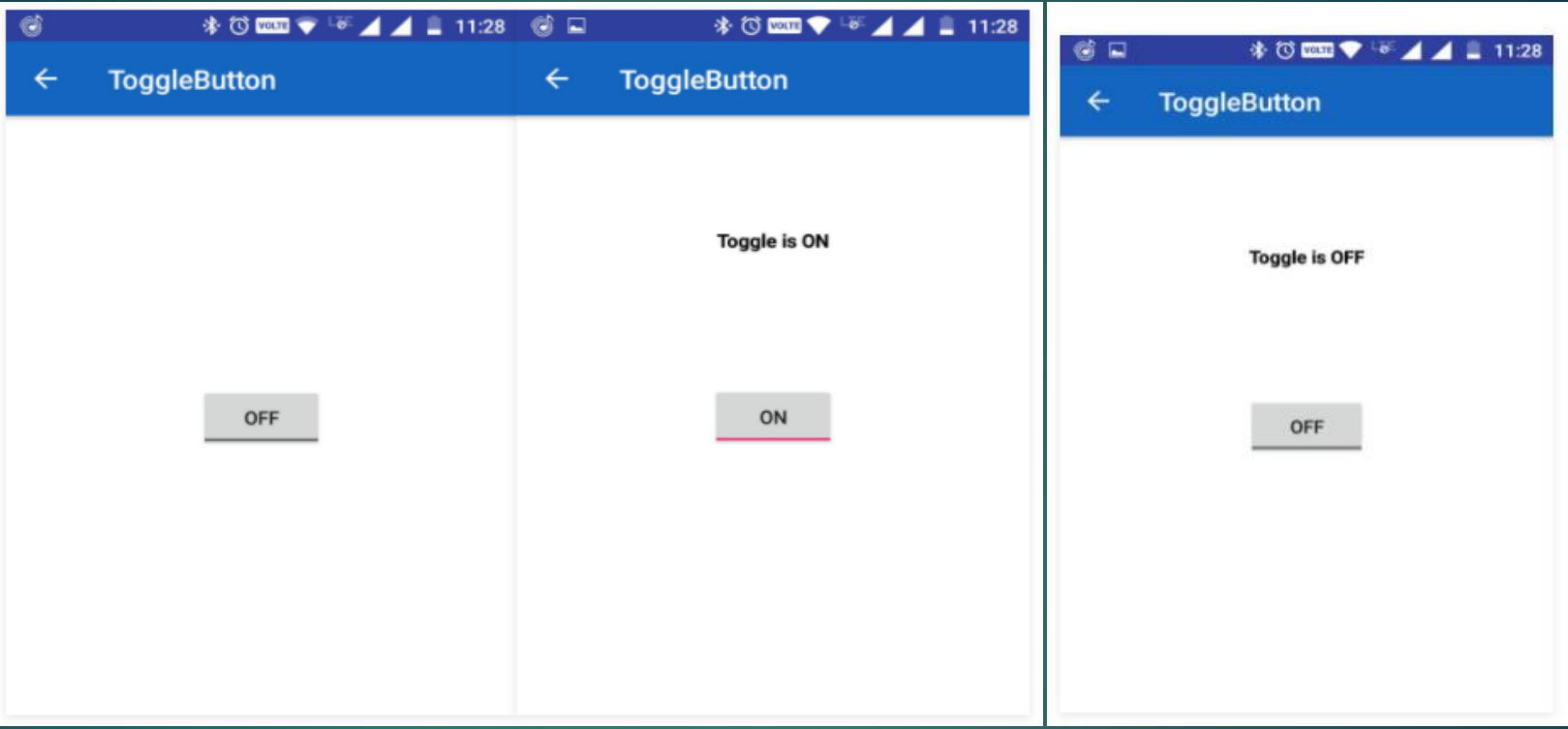
- textOn attribute is used to set the text when toggle button is in checked/on state.

```
android:textOn="Enable"
```

Doing Programmatically:

- simpleToggleButton.setTextOn("Enable");

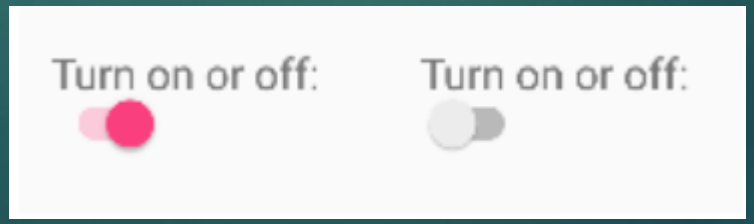
Toggle buttons




ToggleButtonExample
ON
OFF
SUBMIT
ToggleButton1 : ON
ToggleButton2 : OFF

Switches

- Switch is a two-state toggle switch widget that is used to select a choice between two options
- It is used to display checked and unchecked state of a button providing slider control to user.
- Switch is a subclass of CompoundButton.
- It is basically an off/on button which indicate the current state of Switch.
- It is commonly used in selecting on/off in Sound, Bluetooth, WiFi etc.
- As compared to ToggleButton Switch provides user slider control. The user can simply tap on a switch to change its current state.




SwitchExample
switch 1
switch 2
SUBMIT
Switch1 :On
Switch2 :Off

Spinners

- Spinner provides an easy way to select one item from the list of items and it shows a dropdown list of all values when we click on it
- If we have a long list of choices, a spinner might extend beyond layout, forcing the user to scroll. A spinner scrolls automatically, with no extra code needed

Steps :

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

Spinners

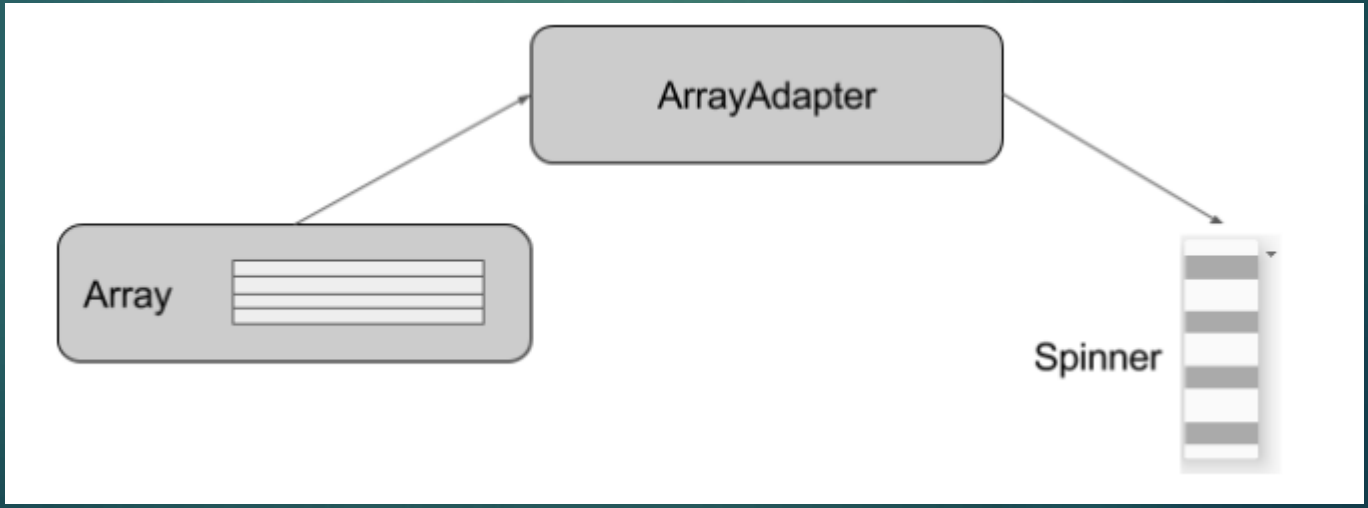
Spinners

- Create the Spinner UI element:

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

Spinners

- Specify values for the Spinner
- Add an adapter that fills the Spinner list with values. An adapter is like a bridge or intermediary, between two incompatible interfaces
- The adapter takes the data set we have specified (an array), and makes a View for each item in the data set (a View within the Spinner)



Spinners

- The SpinnerAdapter class, which implements the Adapter class, allows you to define two different views:
  - one that shows the data values in the Spinner itself,
  - one that shows the data in the drop-down list when the Spinner is touched or clicked
- The values you provide for the Spinner can come from any source, but must be provided through a SpinnerAdapter, such as an ArrayAdapter if the values are easily stored in an array

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

Spinners

- 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

Spinners

- Create the Spinner and its adapter:
- Create the Spinner, and set its listener to the Activity that implements the callback methods.
- add a statement that creates the ArrayAdapter
- Statement of creating adapter includes layout resource that defines how the selected choice appears in the spinner control.
- The simple_spinner_item layout is provided by the platform and is the default layout you should use unless you'd like to define your own layout for the spinner's appearance
- then call setDropDownViewResource(int) to specify the layout the adapter should use to display the list of spinner choices (simple_spinner_dropdown_item is another standard layout defined by the platform)
- Then call setAdapter() to apply the adapter to Spinner

- Add code to respond to Spinner selections

Example1:

```
<Spinner
    android:id="@+id/spinner"
    android:layout_height="50dp"
    android:layout_width="160dp"
    android:layout_marginEnd="10dp"
    android:layout_marginStart="10dp"
    android:layout_marginBottom="10dp"
    android:layout_marginTop="10dp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>
```

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

    String[] countries={"INDIA","US","UK","RUSSIA","CHINA"};
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

//Getting the instance of Spinner and applying OnItemSelectedListener on it
        Spinner spin = (Spinner) findViewById(R.id.spinner);
        spin.setOnItemSelectedListener(this);

//Creating the ArrayAdapter instance having the country name list
        ArrayAdapter aa = new ArrayAdapter
                ( context: this,android.R.layout.simple_spinner_item,countries);
        aa.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
//Setting the ArrayAdapter data on the Spinner
        spin.setAdapter(aa);
    }
```

```
    //Performing action onItemSelected and onNothing selected
    @Override
    public void onItemSelected(AdapterView<?> arg0, View arg1, int position,long id) {
        Toast.makeText(getApplicationContext(), countries[position], Toast.LENGTH_LONG).show();
    }

    @Override
    public void onNothingSelected(AdapterView<?> arg0) {
// TODO Auto-generated method stub

    }
```

Example2:

```
<resources>
    <string name="app_name">Spinner2</string>
    <string-array name="planets_array">
        <item>Mercury</item>
        <item>Venus</item>
        <item>Earth</item>
        <item>Mars</item>
        <item>Jupiter</item>
        <item>Saturn</item>
        <item>Uranus</item>
        <item>Neptune</item>
    </string-array>
</resources>
```

```
<Spinner
    android:id="@+id/spinner"
    android:layout_height="50dp"
    android:layout_width="160dp"
    android:layout_marginEnd="10dp"
    android:layout_marginStart="10dp"
    android:layout_marginBottom="10dp"
    android:layout_marginTop="10dp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>
```

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



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

//Getting the instance of Spinner and applying OnItemSelectedListener on it
        Spinner spin = (Spinner) findViewById(R.id.spinner);
        spin.setOnItemSelectedListener(this);

//Creating the ArrayAdapter instance having the country name list
        ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource( context: this,
                R.array.planets_array, android.R.layout.simple_spinner_item);
        adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
//Setting the ArrayAdapter data on the Spinner
        spin.setAdapter(adapter);
    }
```

```
    //Performing action onItemSelected and onNothing selected
    @Override
    public void onItemSelected(AdapterView<?> parent, View view, int position,long id) {

        String text=parent.getItemAtPosition(position).toString();
        Toast.makeText(parent.getContext(),text, Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onNothingSelected(AdapterView<?> arg0) {
// TODO Auto-generated method stub

    }
```

EditText

- Edittext refers to the widget that displays an empty textfield in which a user can enter the required text
- EditText extends the TextView class, to make the TextView editable
- 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">
    android:hint="@string/enter_text_here"
</EditText>
```

EditText

- Edittext refers to the widget that displays an empty textfield in which a user can enter the required text
- EditText extends the TextView class, to make the TextView editable
- Create an EditText view by adding an EditText to your layout with the following XML:

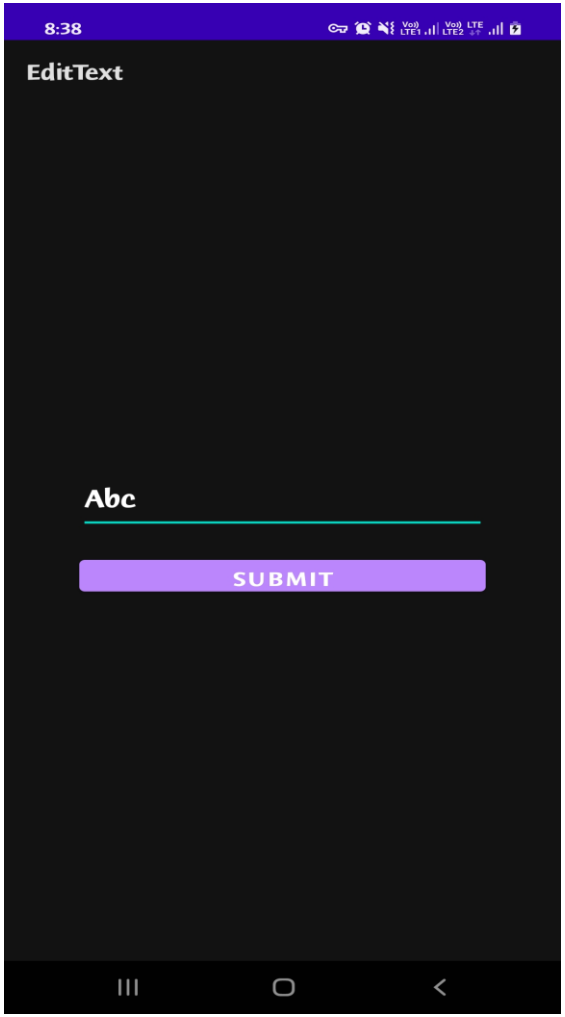
```
<EditText
    android:id="@+id/edit_simple"
    android:layout_height="wrap_content"
    android:layout_width="match_parent">
    android:hint="@string/enter_text_here"
    android:inputType="textShortMessage"
</EditText>
```

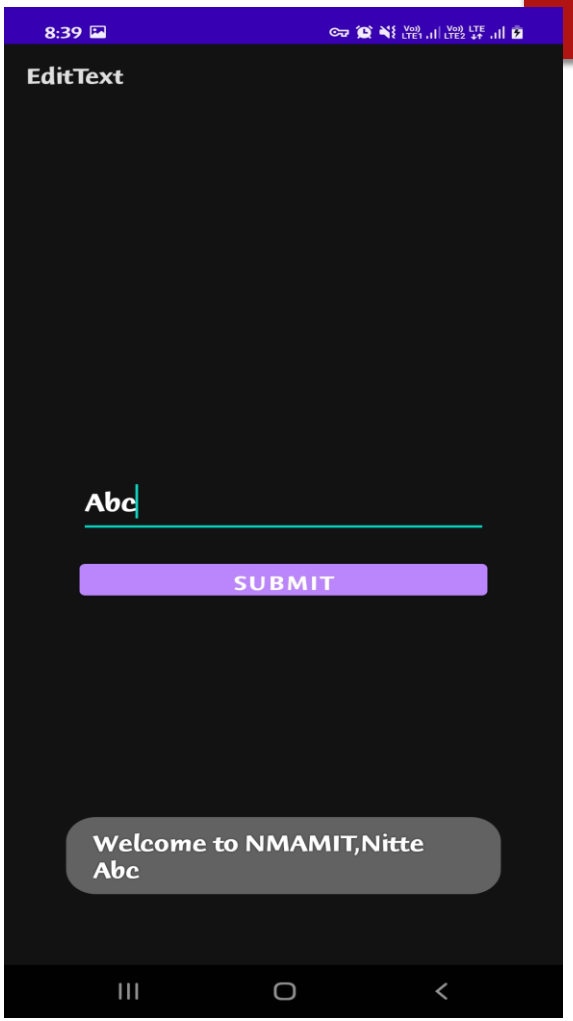
EditText

- The following are generally used attributes for customizing an EditText:
- android:inputType="textCapCharacters": Set the text entry to all capital letters.
- android:inputType="textCapSentences": Start each sentence with a capital letter.
- android:inputType="textMultiLine": Set the text field to enable multiple lines. This value is useful for combining with other attributes. To combine values, concatenate them using the pipe (|) character.
- android:inputType="textPassword": Turn each character the user enters into a dot to conceal an entered password.
- android:inputType="number": Restrict text entry to numbers.
- 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".

```
<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=".MainActivity"
    android:gravity="center">
    <EditText
        android:id="@+id/edittext_id"
        android:layout_width="300dp"
        android:layout_height="60dp"
        android:hint="Enter your Name"/>
    <Button
        android:id="@+id/button_id"
        android:layout_width="300dp"
        android:layout_height="40dp"
        android:layout_below="@+id/edittext_id"
        android:layout_marginTop="20dp"
        android:text="Submit"
        android:textColor="#fff"/>
</RelativeLayout>
```

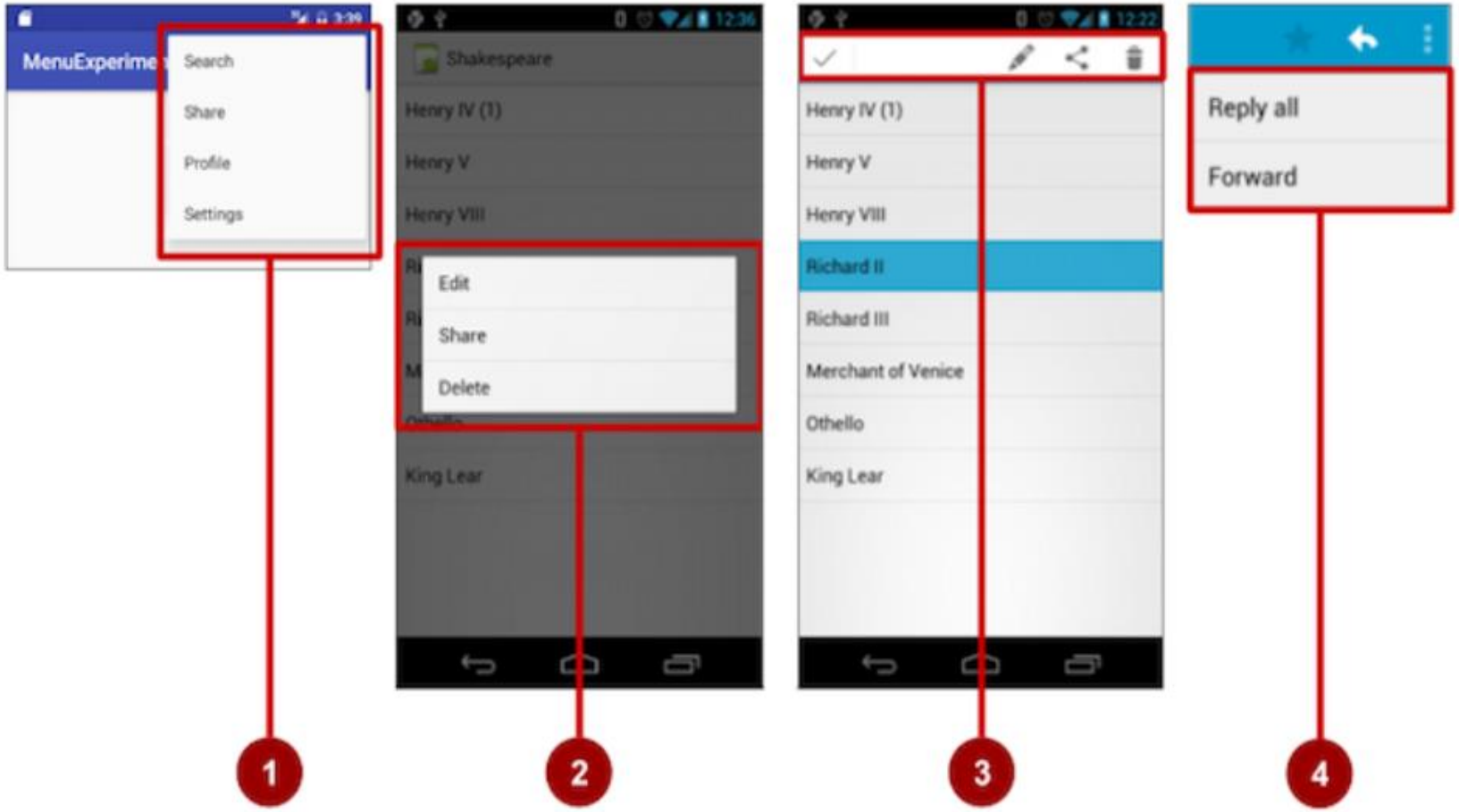
```java
public class MainActivity extends AppCompatActivity {
    private EditText editText;
    private Button button;
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        editText = (EditText)findViewById(R.id.edittext_id);
        button = (Button)findViewById(R.id.button_id);

        button.setOnClickListener(
                new View.OnClickListener() {
                    @Override
                    public void onClick(View v)
                    {
                        String name = editText.getText().toString();
                        Toast.makeText( context: MainActivity.this,
                                text: "Welcome to NMAMIT,Nitte "
                                        + name, Toast.LENGTH_SHORT).show();
                    }
                });
    }
}
```


8:38
EditText
Abc
SUBMIT


8:39
EditText
Abc
SUBMIT
Welcome to NMAMIT,Nitte
Abc

Menus

- In android, Menu is an important part of UI component which is used to provide some common functionality around the application
- The user can select from a menu to perform a function, for example searching for information, saving information, editing information, or navigating to a screen.


MenuExperime
Search
Share
Profile
Settings
Shakespeare
Henry IV (1)
Henry V
Henry VIII
Edit
Share
Delete
King Lear
Henry IV (1)
Henry V
Henry VIII
Richard II
Richard III
Merchant of Venice
Othello
King Lear
Reply all
Forward
1
2
3
4

Types

- Options menu:
- Appears in the app bar and provides the primary options that affect use of the app itself.
- It's where you should place actions that have a global impact on the app.
- Examples of menu options: Search to perform a search, Share to share a link, and Settings to navigate to a Settings Activity

- Contextual menu:
- Appears as a floating list of choices when the user performs a long tap on an element on the screen.
- It provides actions that affect the selected content.
- Examples of menu options: Edit to edit the element, Delete to delete it, and Share to share it over social media

Types

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

- Popup menu:
- Appears anchored to a View such as an ImageButton, and provides an overflow of actions.
- 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/ action bar

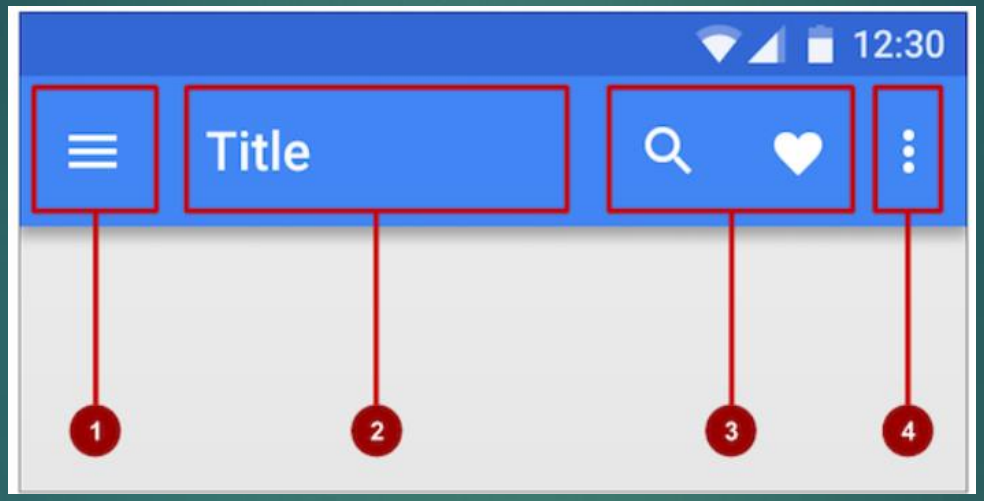
- It is a dedicated space at the top of each Activity screen
- When you create an Activity from a template (such as Empty Template), an app bar is automatically included for the Activity
- The app bar by default shows the app title, or the name defined in AndroidManifest.xml by the android:label attribute for the Activity.
- The app bar may also include the Up button for navigating up to the parent activity

The app bar/ action bar

- The options menu in the app bar usually provides navigation to other screens in the app, or options that affect using the app itself.
- The options menu should not include options that act on an element on the screen. For that a contextual menu is used
- options menu might let the user navigate to another activity to place an order OR might let the user change settings or profile information, or do other actions that have a global impact on the app
- The options menu appears in the right corner of the app bar

The app bar/ action bar

- The app bar is split into four functional areas that apply to most apps



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

Title:

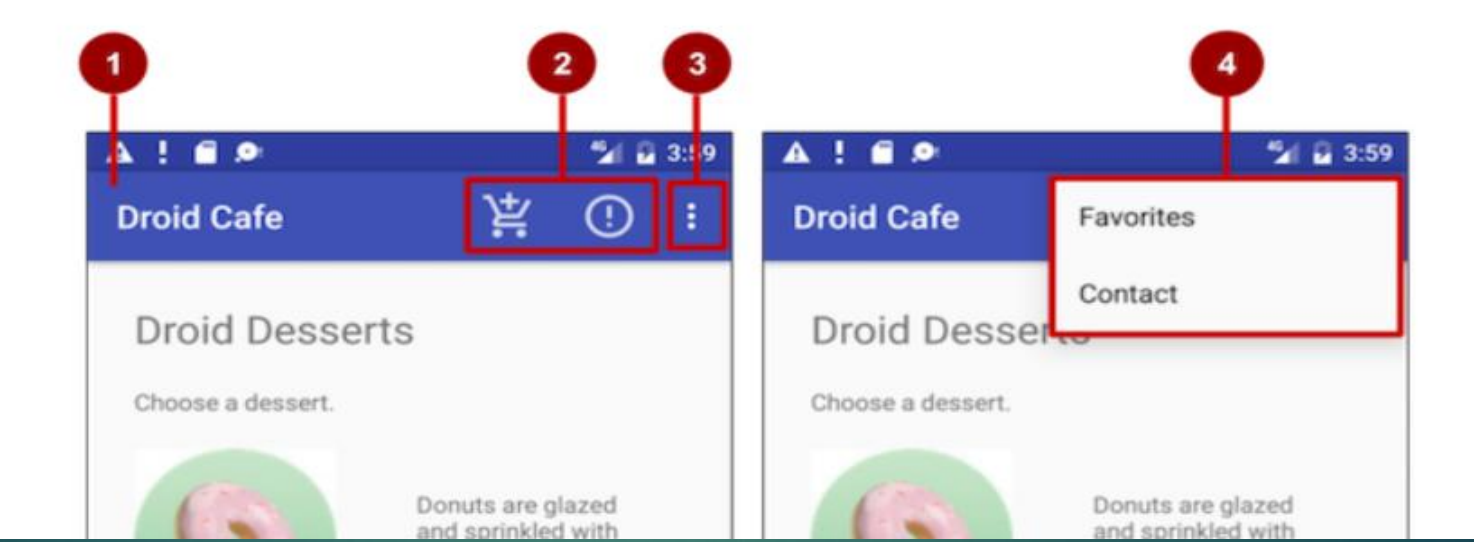
- The title in the app bar is the app title, or the name defined in AndroidManifest.xml by the android:label attribute for the activity

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

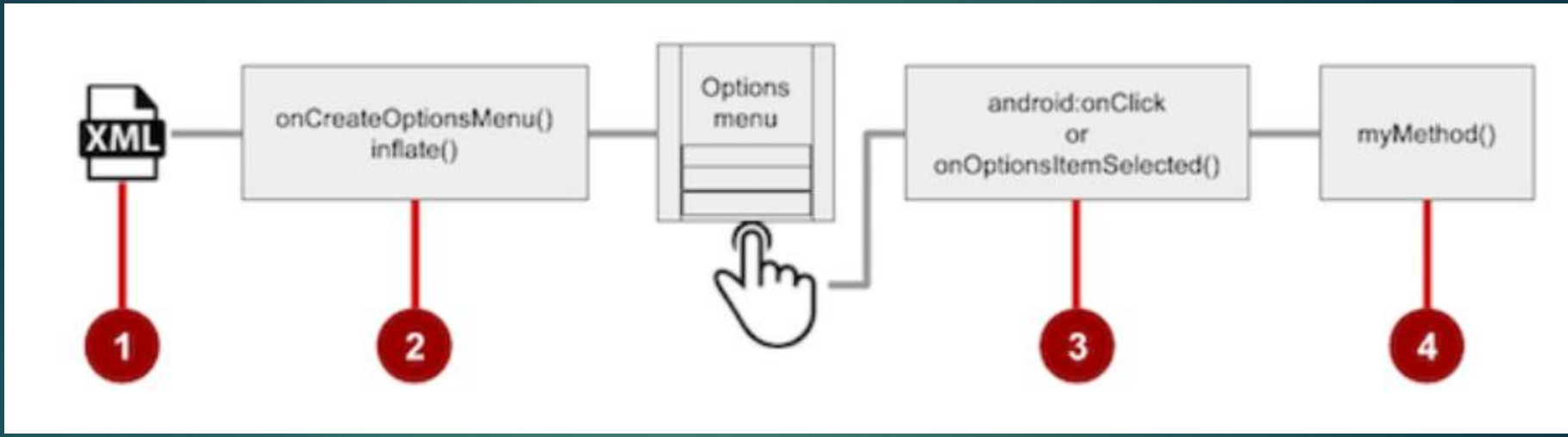
Overflow options menu:

- The overflow icon opens a popup with option menu items that are not shown as icons in the app bar.



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.

Adding the options menu



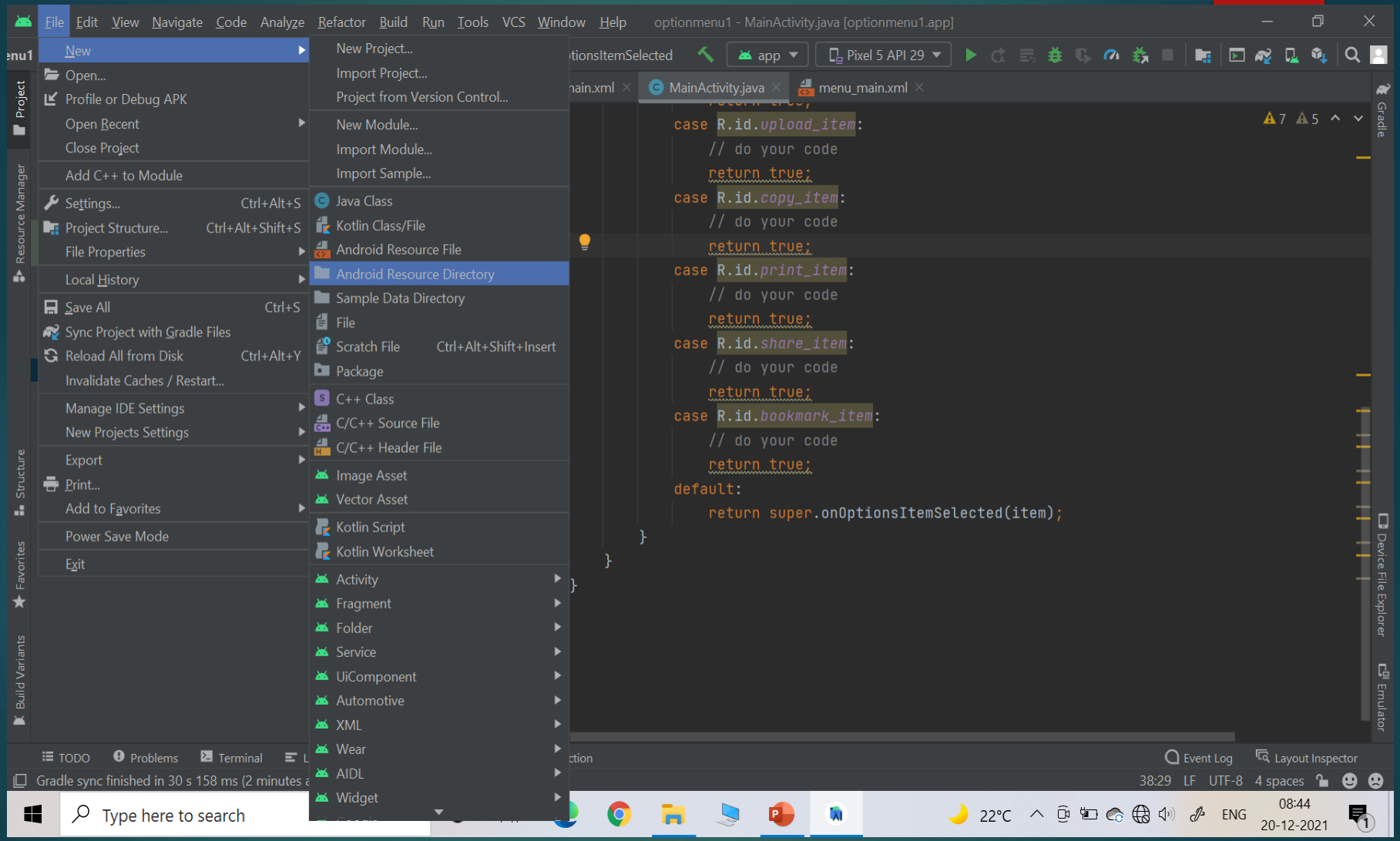
Adding the options menu

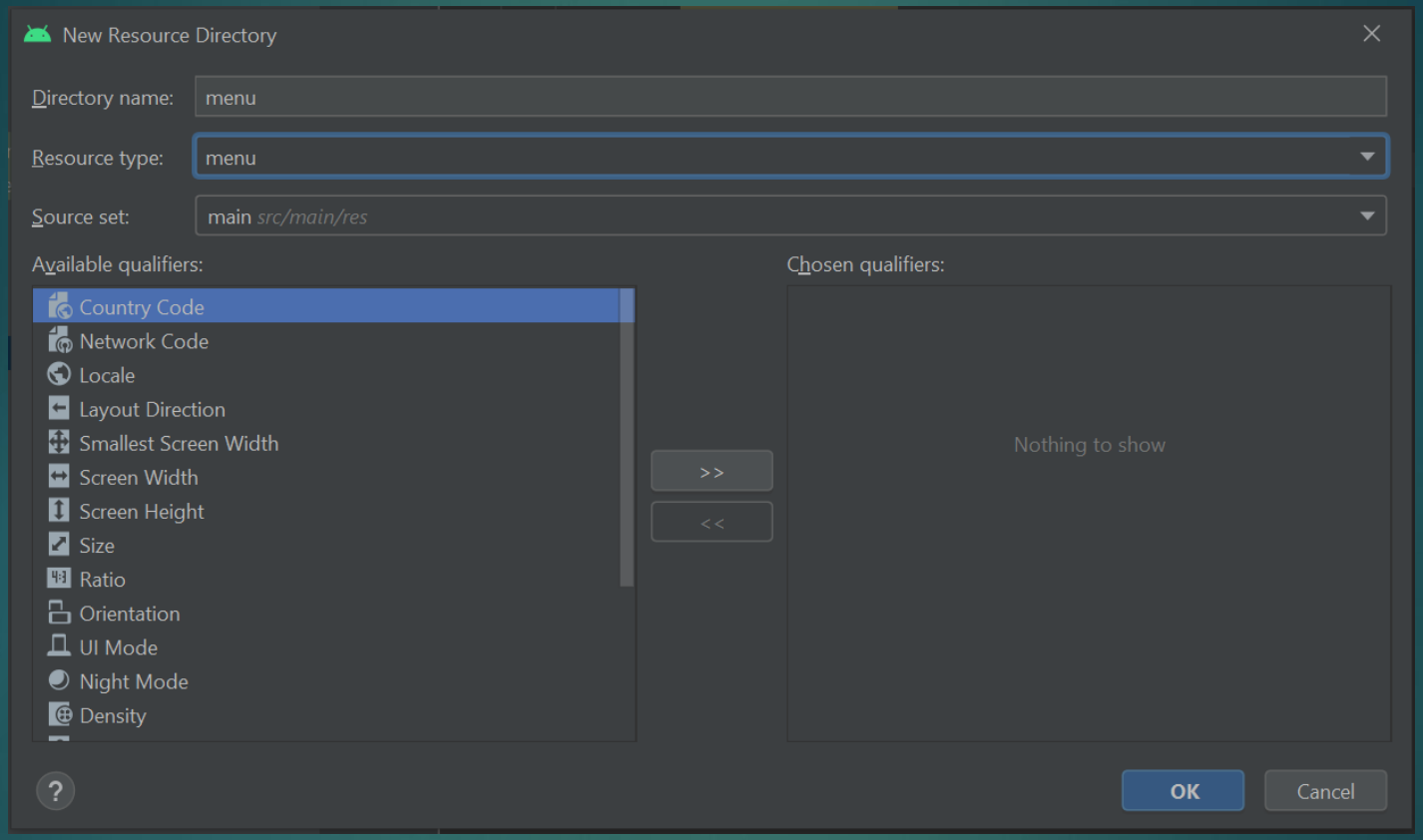
1. XML menu resource: Create an XML menu resource file for the menu items, and assign appearance and position attributes
2. Inflating the menu: Override the onCreateOptionsMenu() method in your Activity to inflate the menu.
3. Handling menu-item clicks: Menu items are View elements, so you can use the android:onClick attribute for each menu item. However, the onOptionsItemSelected() method can handle all the menu-item clicks in one place and determine which menu item the user clicked, which makes your code easier to understand.
4. Performing actions: Create a method to perform an action for each options menu item

Creating an XML resource for the menu

- Follow these steps to add menu items to an XML menu resource:

- Select the res folder in the Project › Android pane and choose File › New › Android resource directory.
- Choose menu in the Resource type drop-down menu, and click OK.
- Select the new menu folder, and choose File › New › Menu resource file.
- Enter the name, such as menu_main, and click OK. The new menu_main.xml file now resides within the menu folder.
- Open menu_main.xml and click the Text tab to show the XML code.
- Add menu items using the ‹item ... /› tag


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
optionmenu1 - MainActivity.java [optionmenu1.app]
New
Open...
Profile or Debug APK
Open Recent
Close Project
Add C++ to Module
Settings... Ctrl+Alt+S
Project Structure... Ctrl+Alt+Shift+S
File Properties
Local History
Save All Ctrl+S
Sync Project with Gradle Files
Reload All from Disk Ctrl+Alt+Y
Invalidate Caches / Restart...
Manage IDE Settings
New Projects Settings
Export
Print...
Add to Favorites
Power Save Mode
Exit
New Project...
Import Project...
Project from Version Control...
New Module...
Import Module...
Import Sample...
Java Class
Kotlin Class/File
Android Resource File
Android Resource Directory
Sample Data Directory
File
Scratch File Ctrl+Alt+Shift+Insert
Package
C++ Class
C/C++ Source File
C/C++ Header File
Image Asset
Vector Asset
Kotlin Script
Kotlin Worksheet
Activity
Fragment
Folder
Service
UiComponent
Automotive
XML
Wear
AIDL
Widget
MainActivity.java
menu_main.xml
case R.id.upload_item:
// do your code
return true;
case R.id.copy_item:
// do your code
return true;
case R.id.print_item:
// do your code
return true;
case R.id.share_item:
// do your code
return true;
case R.id.bookmark_item:
// do your code
return true;
default:
return super.onOptionsItemSelected(item);
TODO Problems Terminal
Event Log Layout Inspector
Gradle sync finished in 30 s 158 ms (2 minutes
38:29 LF UTF-8 4 spaces
Type here to search
22°C
ENG
08:44
20-12-2021


New Resource Directory
Directory name:
menu
Resource type:
menu
Source set:
main src/main/res
Available qualifiers:
Country Code
Network Code
Locale
Layout Direction
Smallest Screen Width
Screen Width
Screen Height
Size
Ratio
Orientation
UI Mode
Night Mode
Density
>>
<<
Chosen qualifiers:
Nothing to show
OK
Cancel

- To add more options menu items, add more ‹item ... /› tags in the xml file
- Within each ‹item ... /› tag, you add attributes that define how the menu item appear
- Whenever possible, show the most-used actions using icons in the app bar so that the user can tap these actions without having to first tap the overflow button

Adding icons for menu items

- To specify icons for actions, first add the icons as image assets to the drawable folder by following the steps below

1. Expand **res** in the Project > Android pane, 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.
4. Edit the name of the icon (for example, **ic_order_white** for the **Order** menu item).
5. Click the clip art image (the Android logo) to select a clip art image as the icon. A page of icons appears. Click the icon you want to use.
6. (Optional) Choose **HOLO_DARK** from the **Theme** drop-down menu. This sets the icon to be white against a dark-colored (or black) background. Click **Next**.
7. Click **Finish** in the Confirm Icon Path dialog.

- Icon and appearance attributes:

```
<item
      android:id="@+id/action_order"
      android:icon="@drawable/ic_order_white"
      android:title="@string/action_order"/>
```

- Position attributes
- Use the android:orderInCategory attribute to specify the order in which the menu items appear in the menu, with the lowest number appearing higher in the menu

- Use the app:showAsAction attribute to show menu items as icons in the app bar, with the following values
- "always": Always place this item in the app bar. Use this only if it's critical that the item appear in the app bar (such as a Search icon). If you set multiple items to always appear in the app bar, they might overlap something else in the app bar, such as the app title.
- "ifRoom": Only place this item in the app bar if there is room for it. If there is not enough room for all the items marked "ifRoom", the items with the lowest orderInCategory values are displayed in the app bar. The remaining items are displayed in the overflow menu.
- "never": Never place this item in the app bar. Instead, list the item in the app bar's overflow menu.
- "withText": Also include the title text (defined by android:title) with the item. This attribute is used primarily to include the title with the icon in the app bar, because if the item appears in the overflow menu, the title text appears regardless

- You can add a submenu to an item in any menu by adding a ‹menu› element as the child of an ‹item›

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/file"
          android:title="@string/file" >
        <!-- "file" submenu -->
        <menu>
            <item android:id="@+id/create_new"
                  android:title="@string/create_new" />
            <item android:id="@+id/open"
                  android:title="@string/open" />
        </menu>
    </item>
</menu>
```

Inflating the menu resource

- inflate the menu resource in your activity by overriding the onCreateOptionsMenu() method and using the getMenuInflater() method of the Activity class
- getMenuInflater() method returns a MenuInflater, which is a class used to instantiate menu XML files into Menu objects
- The MenuInflater class provides the inflate() method, which takes two parameters:
  - The resource id for an XML layout resource to load
  - The Menu to inflate into

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
 getMenuInflater().inflate(R.menu.menu_main, menu);
 return true;
}
```

Handling the menu-item click

- onOptionsItemSelected() method can handle all the menu-item clicks in one place and determine which menu item the user clicked

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_order:
            showOrder();
            return true;
        case R.id.action_status:
            showStatus();
            return true;
        case R.id.action_contact:
            showContact();
            return true;
        default:
            // Do nothing
    }
    return super.onOptionsItemSelected(item);
}
```

menu_main.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:id="@+id/search_item"
        android:title="Search" />
    <item android:id="@+id/upload_item"
        android:title="Upload" />
    <item android:id="@+id/copy_item"
        android:title="Copy" />
    <item android:id="@+id/print_item"
        android:title="Print" />
    <item android:id="@+id/share_item"
        android:title="Share" />
    <item android:id="@+id/bookmark_item"
        android:title="BookMark" />
</menu>
```

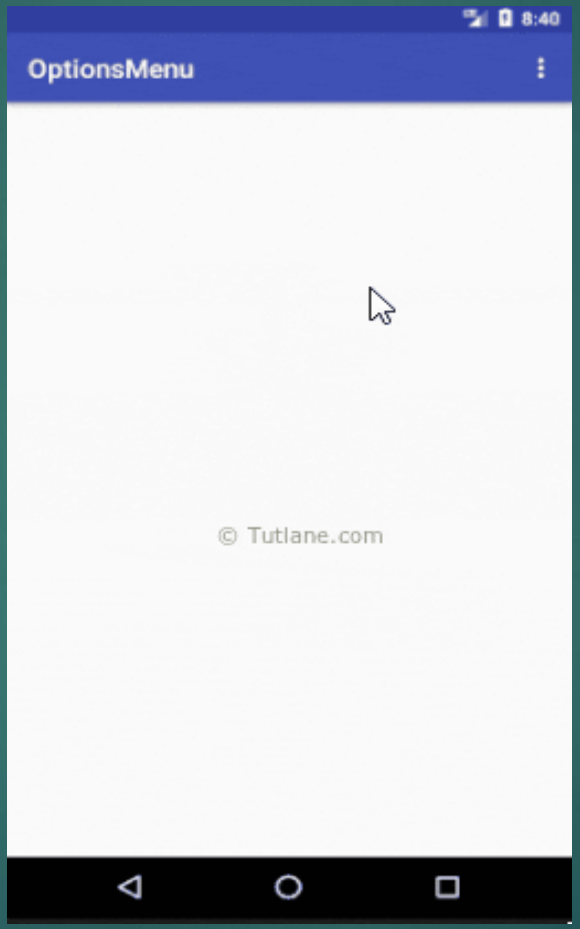
MainActivity.java

```java
package com.example.optionmenu1;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;


public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }
```

MainActivity.java

```java
public boolean onOptionsItemSelected(MenuItem item) {
    Toast.makeText( context: this,  text: "Selected Item: " +item.getTitle(), Toast.LENGTH_SHORT).show();
    switch (item.getItemId()) {
        case R.id.search_item:
            // do your code
            return true;
        case R.id.upload_item:
            // do your code
            return true;
        case R.id.copy_item:
            // do your code
            return true;
        case R.id.print_item:
            // do your code
            return true;
        case R.id.share_item:
            // do your code
            return true;
        case R.id.bookmark_item:
            // do your code
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

MainActivity.java



Contextual Menu

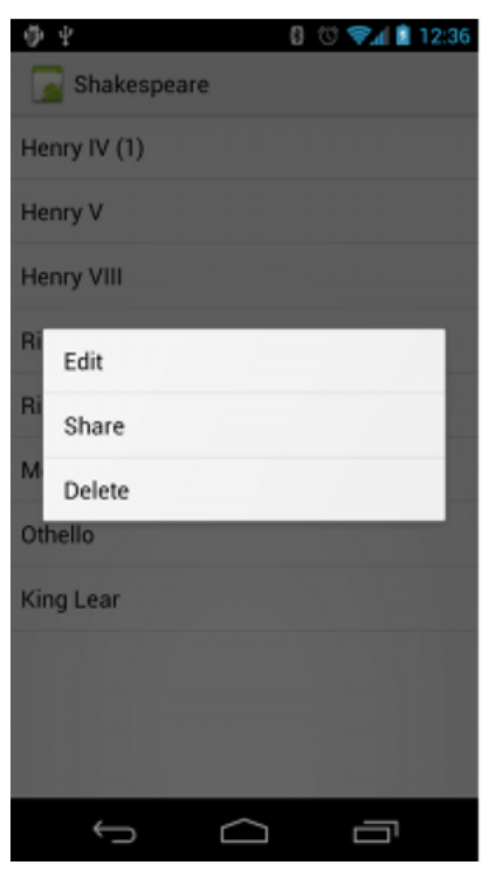
- We use a contextual menu to allow users to take an action on a selected View
- It appears when the user performs a long press or click on an element

Android provides two kinds of contextual menus:

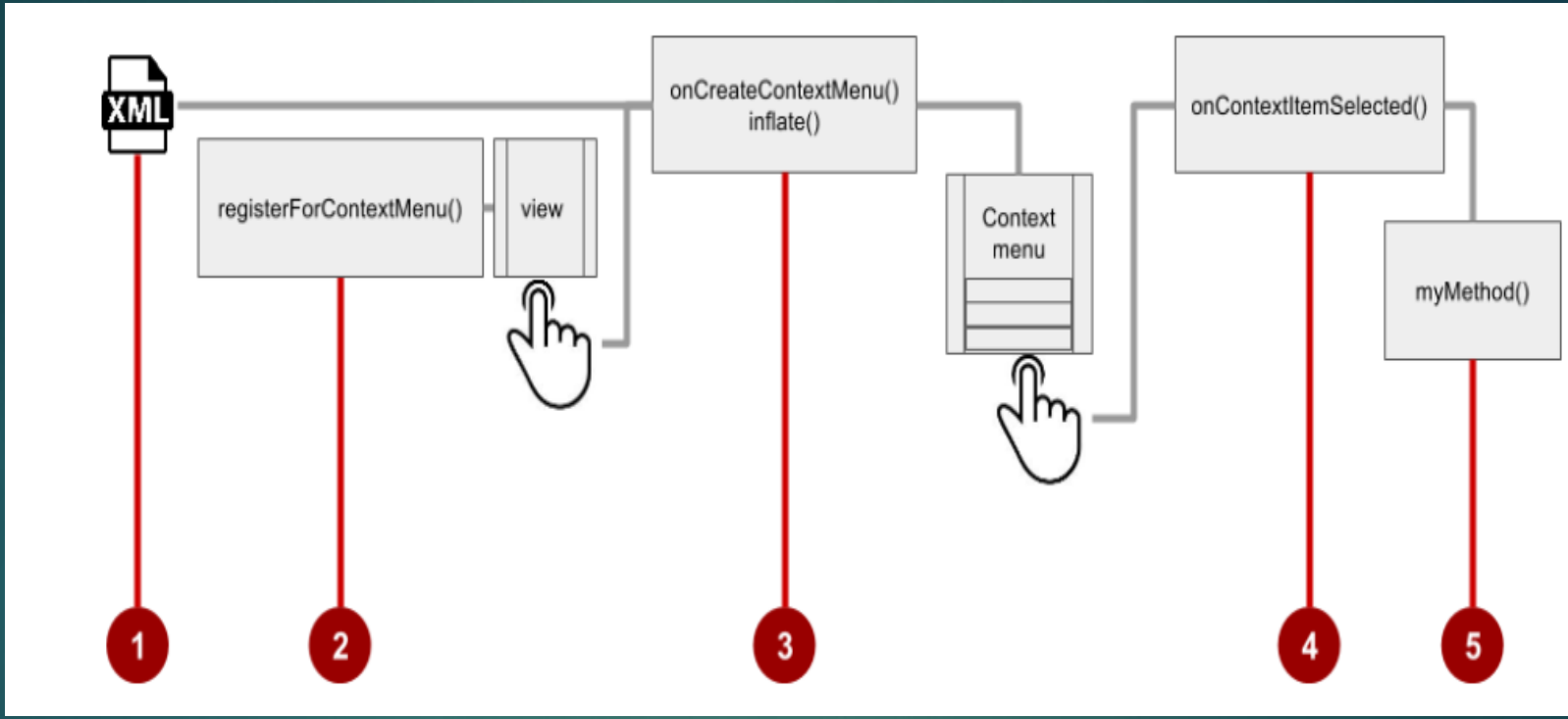
Floating context menu:

- A menu appears as a floating list of menu items when the user performs a long-click (press and hold) on a view that declares support for a context menu.
- Users can perform a contextual action on one item at a time


SHAKESPEARES
Image result for Shakespeare
Download image
Open image in new tab
Search Google for this image
Share image
William Shakespeare - Wikipedia


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

Steps:


XML
registerForContextMenu()
view
onCreateContextMenu()
inflate()
Context
menu
onContextItemSelected()
myMethod()
1
2
3
4
5

Contextual Menu

- Create an XML menu resource file for the menu items
- Register a View to the context menu using the registerForContextMenu() method of the Activity class.
- Implement the onCreateContextMenu() method in your Actvity to inflate the menu
- Implement the onContextItemSelected() method in your Activity to handle menu-item clicks
- Create a method to perform an action for each context menu item

Creating the XML resource file

```
<item
    android:id="@+id/context_edit"
    android:title="Edit"
    android:orderInCategory="10"/>
```

Registering a View to the context menu

- To register a View to the context menu, call the registerForContextMenu() method with the View
- Registering a context menu for a view sets the View.OnCreateContextMenuListener on the View to this activity, so that onCreateContextMenu() is called when it's time to show the context menu

```
// Registering the context menu to the TextView of the article.
TextView article_text = findViewById(R.id.article);
registerForContextMenu(article_text);
```

Implementing the onCreateContextMenu() method

- When the registered View receives a long-click event, the system calls the onCreateContextMenu() method, which we can override in our Activity
- The onCreateContextMenu() method is where we define the menu items, usually by inflating a menu resource

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
                                ContextMenu.ContextMenuInfo menuInfo) {
   super.onCreateContextMenu(menu, v, menuInfo);
   MenuInflater inflater = getMenuInflater();
   inflater.inflate(R.menu.menu_context, menu);
}
```

Implementing the onContextItemSelected() method

- When the user clicks on a menu item, the system calls the onContextItemSelected() method
- We override this method in our Activity in order to determine which menu item was clicked, and for which view the menu is appearing

```
@Override
public boolean onContextItemSelected(MenuItem item) {
   switch (item.getItemId()) {
      case R.id.context_edit:
         editNote();
         return true;
      case R.id.context_share:
         shareNote();
         return true;
      default:
         return super.onContextItemSelected(item);
   }
}
```

context_menu.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/option1"
        android:title="Option1" />
    <item android:id="@+id/option2"
        android:title="Option2" />
    <item android:id="@+id/option3"
        android:title="Option3" />
</menu>
```

activity_main.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.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=".MainActivity">

    <TextView
        android:id="@+id/text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

main_activity.java

```java
package com.example.contextmenu;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.ContextMenu;
import android.view.MenuInflater;
import android.view.View;
import android.widget.TextView;
import android.view.MenuItem;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        TextView textView=findViewById(R.id.text_view);
        registerForContextMenu(textView);
    }
    public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo menuInfo)
    {
        super.onCreateContextMenu(menu, v, menuInfo);
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.context_menu, menu);
        menu.setHeaderTitle("Select The Action");
    }
```

main_activity.java

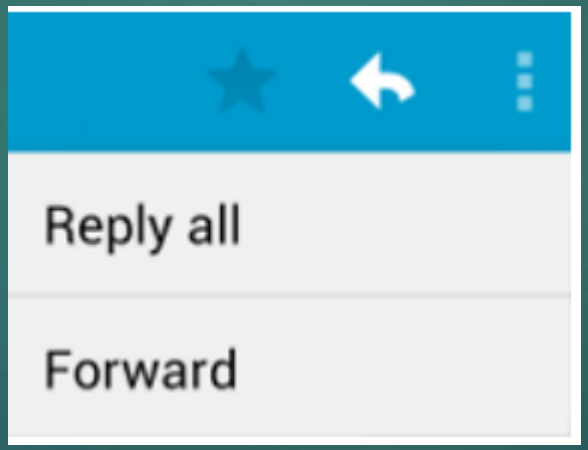
```java
public boolean onContextItemSelected(MenuItem item) {

    switch (item.getItemId()) {
        case R.id.option1:

            Toast.makeText( context: this,  text: "Option1 selected", Toast.LENGTH_SHORT).show();
            return true;
        case R.id.option2:
            Toast.makeText( context: this,  text: "Option2 selected", Toast.LENGTH_SHORT).show();
            return true;
        case R.id.option3:
            Toast.makeText( context: this,  text: "Option3 selected", Toast.LENGTH_SHORT).show();
            return true;

        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Popup menu

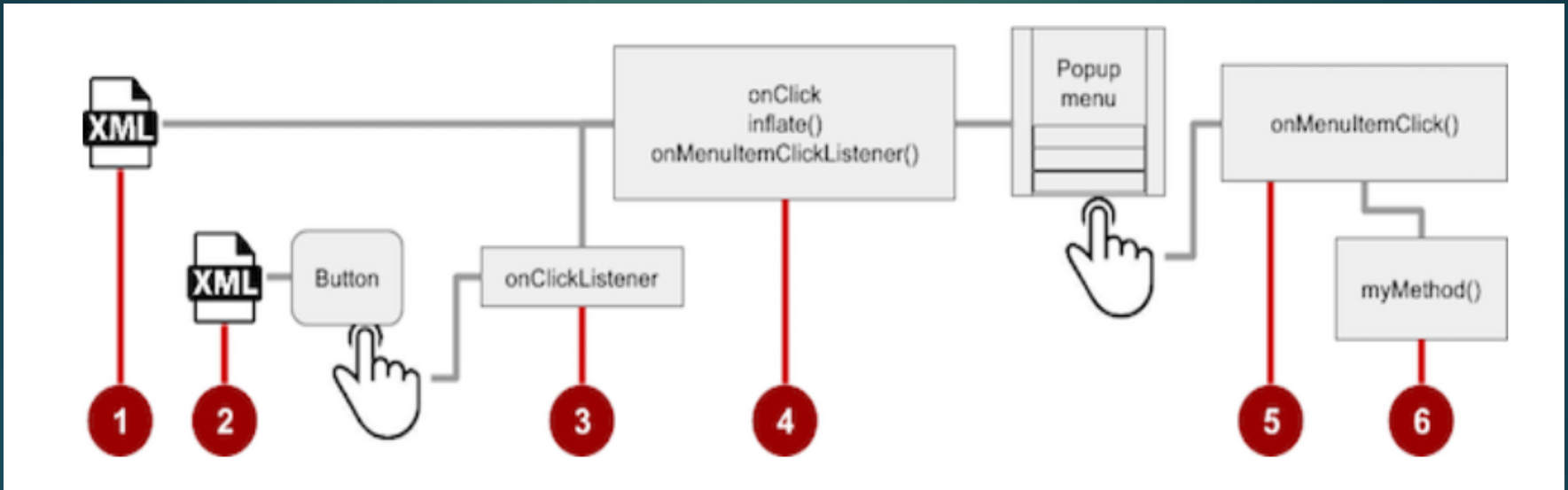
- A PopupMenu is a vertical list of items anchored to a View.
- It appears below the anchor View if there is room, or above the View otherwise
- A popup menu is typically used to provide an overflow of actions
- Unlike a context menu, a popup menu is anchored to a Button, is always available, and its actions generally do not affect the content of the View

Popup menu

- Example: Gmail app uses a popup menu anchored to the overflow icon in the app bar when showing an email message
- The popup menu items Reply, Reply All, and Forward are related to the email message, but don't affect or act on the message
- Actions in a popup menu should not directly affect the corresponding content



Steps


XML
XML
Button
onClick
inflate()
onMenuItemClickListener()
onClickListener
Popup
menu
onMenuItemClick()
myMethod()
1
2
3
4
5
6

Popup menu

1. Create an XML menu resource file for the popup menu items
2. Add an ImageButton for the popup menu icon in the XML activity layout file.
3. Assign onClickListener() to the ImageButton.
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.

Creating the XML resource file

- Create the XML menu resource directory and file and Use a suitable name for the file, such as menu_popup

Adding an ImageButton for the icon to click:

- Use an ImageButton in the Activity layout for the icon that triggers the popup menu

```
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/button_popup"
    android:src="@drawable/@drawable/ic_action_popup"/>
```

Assigning onClickListener to the button

- Create a member variable (mButton) in the Activity class definition:
- In the onCreate() method for the same Activity, assign onClickListener() to the ImageButton

```
// ... Rest of Activity code
@Override
protected void onCreate(Bundle savedInstanceState) {
    // ... Rest of onCreate code
    mButton = (ImageButton) findViewById(R.id.button_popup);
    mButton.setOnClickListener(new View.OnClickListener() {
        // Define onClick here ...
    });
}
```

Inflating the popup menu

- As part of the setOnClickListener() method within onCreate(), add the onClick() method to inflate the popup menu and register it with PopupMenu.OnMenuItemClickListener:

```
// Define onClick here ...
@Override
public void onClick(View v) {
    // Create the instance of PopupMenu.
    PopupMenu popup = new PopupMenu(MainActivity.this, mButton);
    // Inflate the Popup using XML file.
    popup.getMenuInflater().inflate(R.menu.menu_popup, popup.getMenu());
    // Register the popup with OnMenuItemClickListener.
    popup.setOnMenuItemClickListener(new
                                PopupMenu.OnMenuItemClickListener() {
        // Add onMenuItemClick here...
        // Perform action here ...
}
```

Implementing onMenuItemClick

- To perform an action when the user selects a popup menu item, implement the onMenuItemClick() callback within te above setOnClickListener() method

```
    // Add onMenuItemClick here...
    public boolean onMenuItemClick(MenuItem item) {
        // Perform action here ...
        return true;
    }
});
// Show the popup menu.
popup.show();
```

popup_menu.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/option1"
        android:title="Option 1" />
    <item android:id="@+id/option2"
        android:title="Option 2" />
    </menu>
```

activity_main.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.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=".MainActivity">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/button"
        android:text="Click"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

MainActivity.java

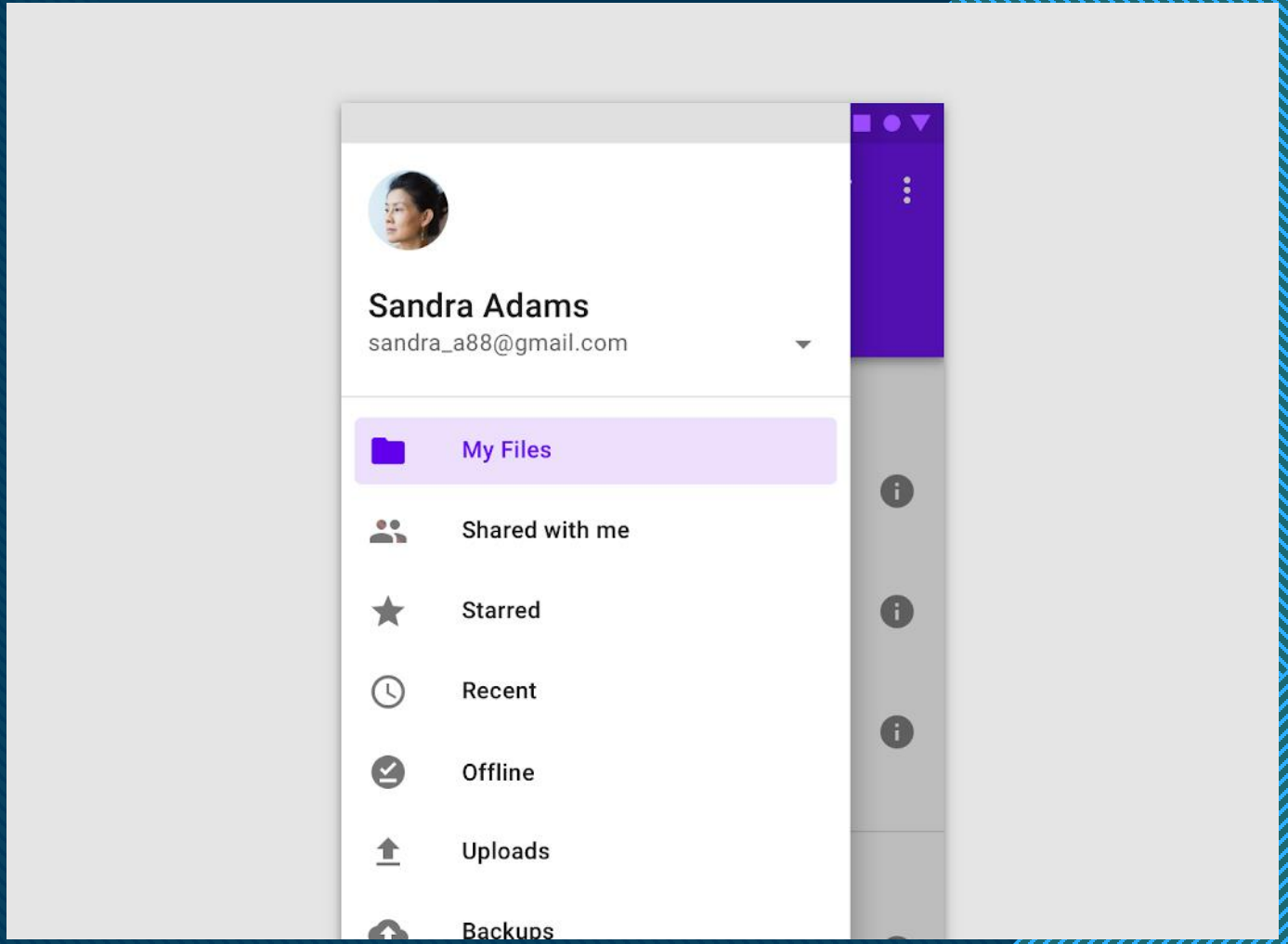
```java
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    button = (Button) findViewById(R.id.button);
    button.setOnClickListener(new View.OnClickListener() {

        @Override
        public void onClick(View v) {
            //Creating the instance of PopupMenu
            PopupMenu popup = new PopupMenu( context: MainActivity.this, button);
            //Inflating the Popup using xml file
            popup.getMenuInflater().inflate(R.menu.popup_menu, popup.getMenu());

            //registering popup with OnMenuItemClickListener
            popup.setOnMenuItemClickListener(new PopupMenu.OnMenuItemClickListener() {
                public boolean onMenuItemClick(MenuItem item) {
                    Toast.makeText( context: MainActivity.this, text: "You Clicked : " + item.getTitle(),
                            Toast.LENGTH_SHORT).show();
                    return true;
                }
            });
            popup.show();//showing popup menu
        }
    });//closing the setOnClickListener method
}
```

Navigation Drawer

By : Sharath Rao A

What is Navigation Drawer

THIS

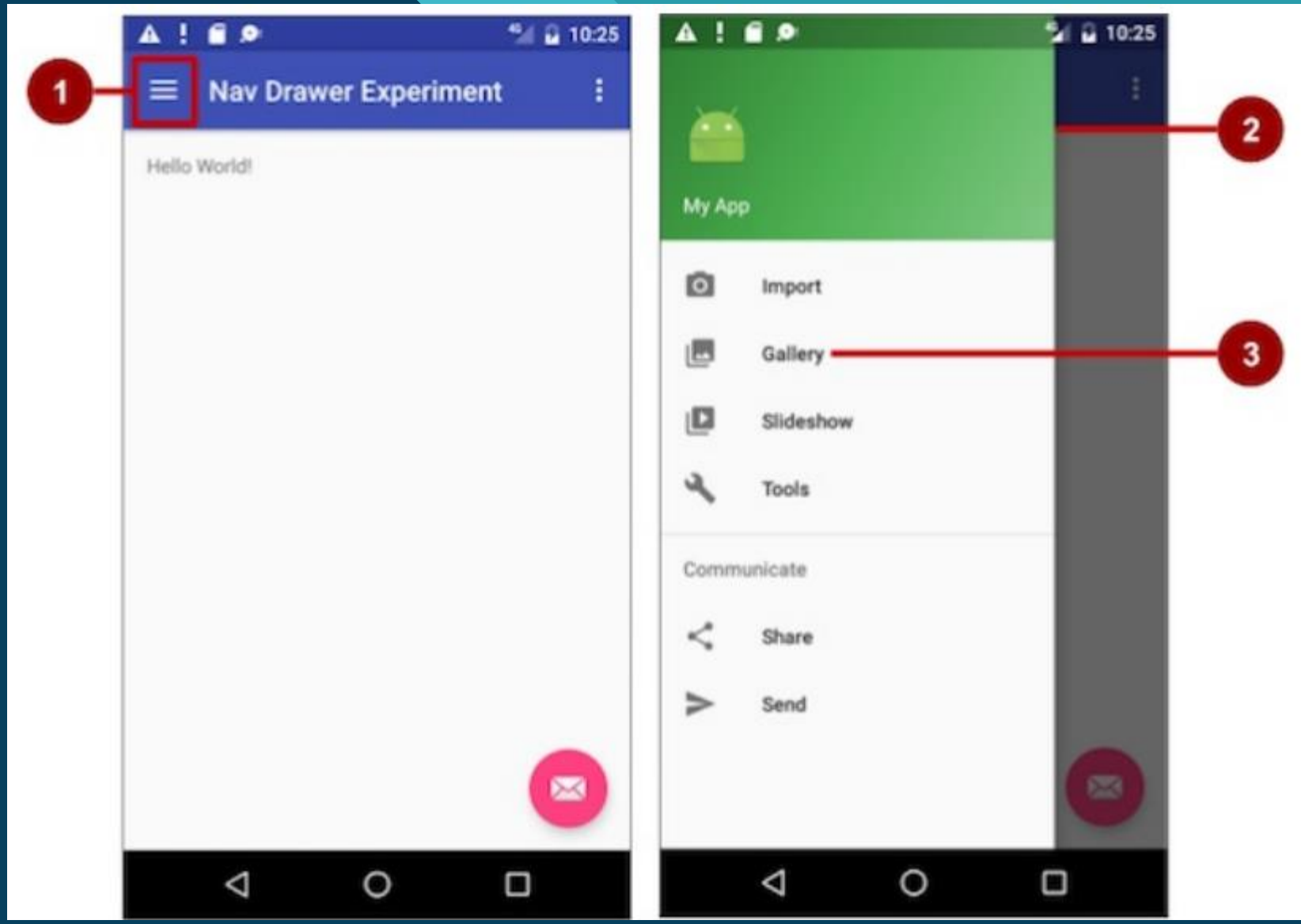


Usage

- A navigation drawer is a panel that usually displays navigation options on the left edge of the screen. It is hidden most of the time, but is revealed when the user swipes a finger from the left edge of the screen or touches the navigation icon in the app bar.

- A good example of a navigation drawer is in the Gmail app, which provides access to the Inbox, labelled email folders, and settings. The best practice for employing a navigation drawer is to provide descendant navigation from the parent activity to all of the other activities or fragments in an app. It can display many navigation targets at once

- 1. Navigation icon in the app bar
- 2. Navigation drawer
- 3. Navigation drawer menu item



How to create Navigation Bar

- 1. Create the following layouts:

  A navigation drawer as the activity layout's root view.

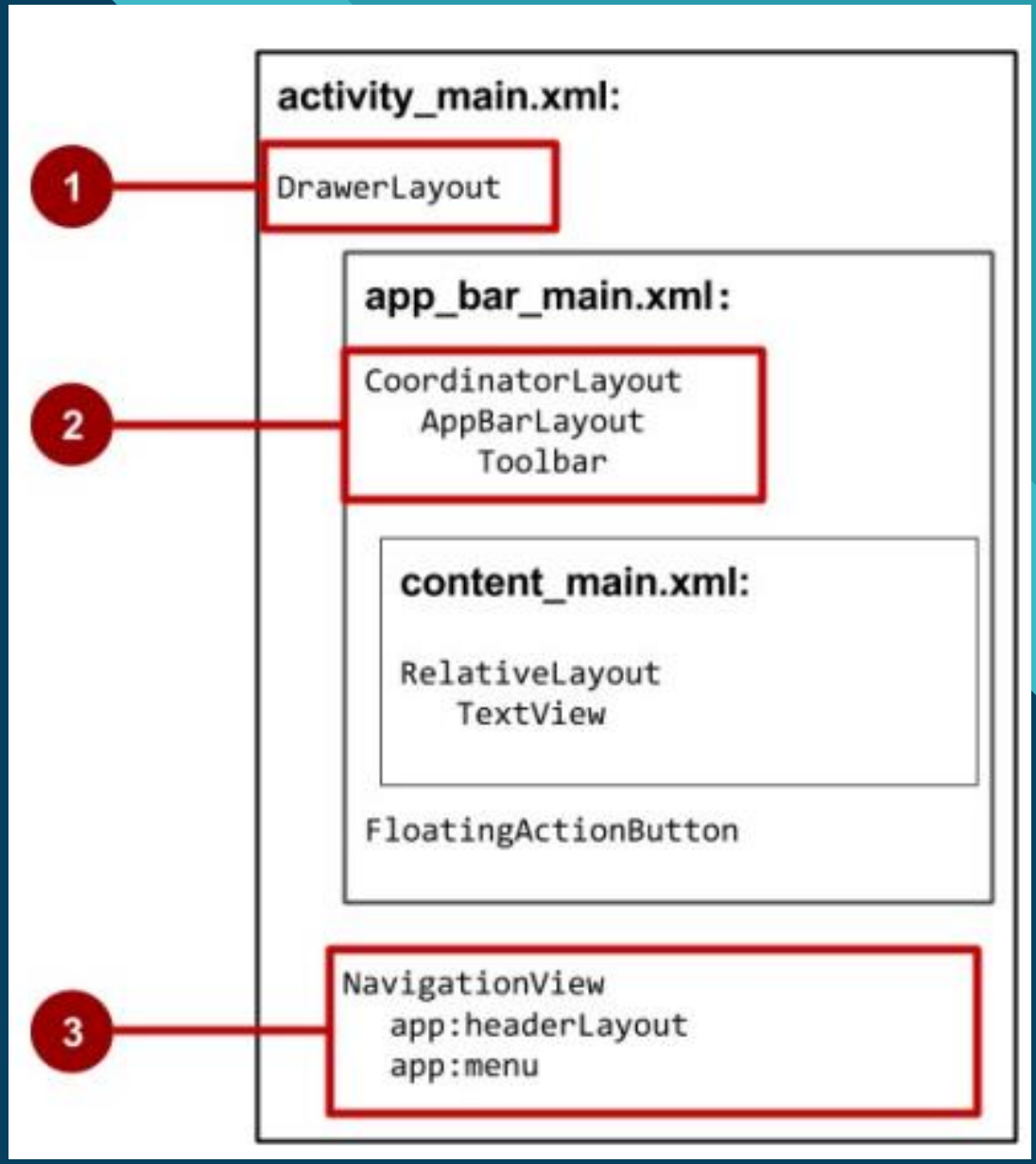
  A navigation view for the drawer itself.

  An app bar layout that will include a navigation icon button.

  A content layout for the activity that displays the navigation drawer.

  A layout for the navigation drawer header.
- 2. Populate the navigation drawer menu with item titles and icons.
- 3. Set up the navigation drawer and item listeners in the activity code.
- 4. Handle the navigation menu item selections

- 1. DrawerLayout is the root view of the activity's layout.
- 2. The included app_bar_main (app_bar_main.xml) uses a CoordinatorLayout as its root, and defines the app bar layout with a Toolbar which will include the navigation icon to open the drawer.
- 3. The NavigationView defines the navigation drawer layout and its header, and adds menu items to it



Populating the navigation drawer menu

- The NavigationView in the activity_main layout specifies the menu items for the navigation drawer using the following statement:
- app:menu="@menu/activity_main_drawer"

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

    <group android:checkableBehavior="none">
        <item
            android:id="@+id/nav_camera"
            android:icon="@drawable/ic_menu_camera"
            android:title="@string/import_camera" />
        <item
            android:id="@+id/nav_gallery"
            android:icon="@drawable/ic_menu_gallery"
            android:title="@string/gallery" />
        <item
            android:id="@+id/nav_slideshow"
            android:icon="@drawable/ic_menu_slideshow"
            android:title="@string/slideshow" />
        <item
            android:id="@+id/nav_manage"
            android:icon="@drawable/ic_menu_manage"
            android:title="@string/tools" />
    </group>

    <item android:title="@string/communicate">
        <menu>
            <item
                android:id="@+id/nav_share"
                android:icon="@drawable/ic_menu_share"
                android:title="@string/share" />
            <item
                android:id="@+id/nav_send"
                android:icon="@drawable/ic_menu_send"
                android:title="@string/send" />
        </menu>
    </item>

</menu>
```

Setting up the navigation drawer and item listeners

To use a listener for the navigation drawer's menu items, the activity hosting the navigation drawer must implement the OnNavigationItemSelectedListener interface:

1. Implement `NavigationView.OnNavigationItemSelectedListener` in the class definition:

```
public class MainActivity extends AppCompatActivity
                implements NavigationView.OnNavigationItemSelectedListener {
...
}
```

This interface offers the onNavigationItemSelected() method, which is called when an item in the navigation drawer menu item is tapped. As you enter `OnNavigationItemSelectedListener` , the red warning bulb appears on the left margin.

Before setting up the navigation item listener, add code to the activity's `onCreate()` method to instantiate the DrawerLayout and NavigationView objects (`drawer` and `navigationView` in the code below):

```
@Override
protected void onCreate(Bundle savedInstanceState) {
   . . .
   DrawerLayout drawer = (DrawerLayout)
                                 findViewById(R.id.drawer_layout);
   ActionBarDrawerToggle toggle =
               new ActionBarDrawerToggle(this, drawer, toolbar,
               R.string.navigation_drawer_open,
               R.string.navigation_drawer_close);
   if (drawer != null) {
      drawer.addDrawerListener(toggle);
   }
   toggle.syncState();

   NavigationView navigationView = (NavigationView)
               findViewById(R.id.nav_view);
   if (navigationView != null) {
      navigationView.setNavigationItemSelectedListener(this);
   }
}
```

The above code instantiates an ActionBarDrawerToggle, which substitutes a special drawable for the activity's **Up** button in the app bar, and links the activity to the DrawerLayout. The special drawable appears as a "hamburger" navigation icon when the drawer is closed, and animates into an arrow as the drawer opens.

```
@Override
public boolean onNavigationItemSelected(MenuItem item) {
   DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
   // Handle navigation view item clicks here.
   switch (item.getItemId()) {
      case R.id.nav_camera:
         // Handle the camera import action (for now display a toast).
         drawer.closeDrawer(GravityCompat.START);
         displayToast(getString(R.string.chose_camera));
         return true;
      case R.id.nav_gallery:
         // Handle the gallery action (for now display a toast).
         drawer.closeDrawer(GravityCompat.START);
         displayToast(getString(R.string.chose_gallery));
         return true;
      case R.id.nav_slideshow:
         // Handle the slideshow action (for now display a toast).
         drawer.closeDrawer(GravityCompat.START);
         displayToast(getString(R.string.chose_slideshow));
         return true;
      case R.id.nav_manage:
         // Handle the tools action (for now display a toast).
         drawer.closeDrawer(GravityCompat.START);
         displayToast(getString(R.string.chose_tools));
         return true;
```

Subtitle

Thank You

RecyclerView

- RecyclerView makes it easy to efficiently display large sets of data
- The RecyclerView class is a more advanced and flexible version of ListView.
- It's a container for displaying large, scrollable data sets efficiently by maintaining a limited number of View items.

RecyclerView components

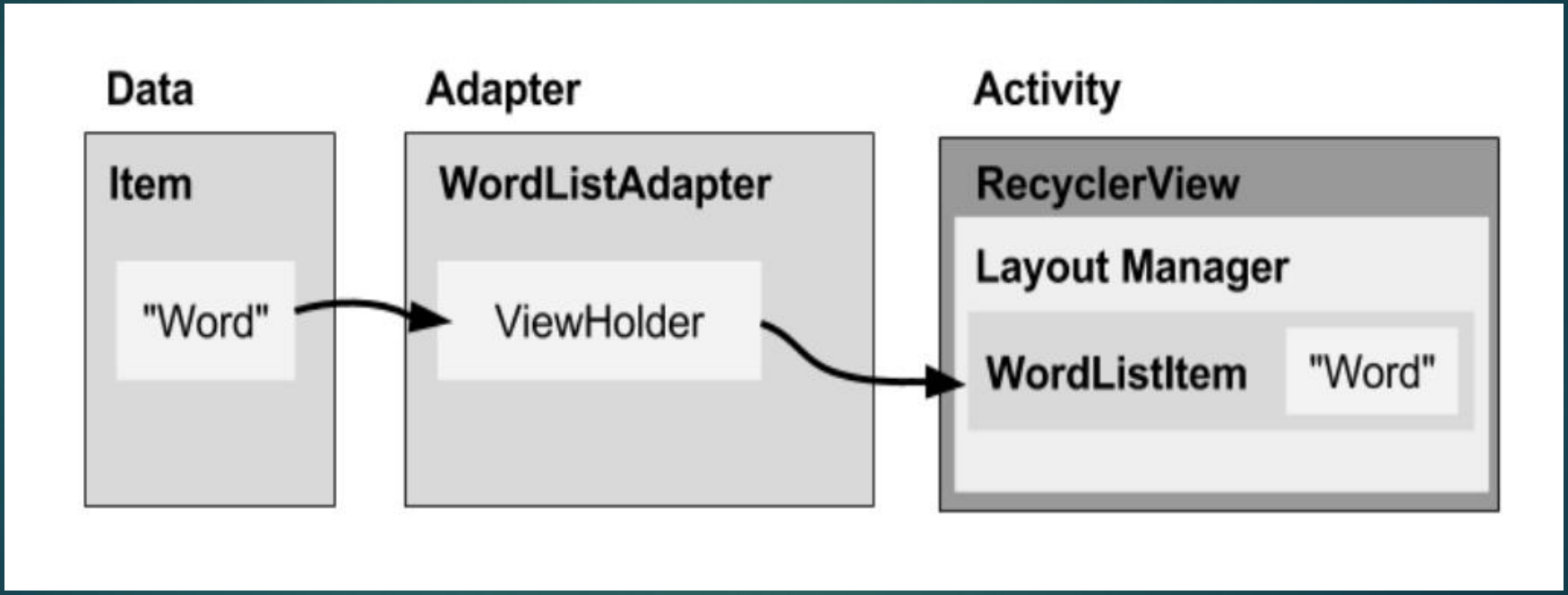
- To display data in a RecyclerView, you need the following
- Data:
  - can be taken from database, cloud or locally
- A RecyclerView:
  - The scrolling list that contains the list items.
  - An instance of RecyclerView as defined in the Activity layout file to act as the container for the View items
- 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 layout, so that one View item at a time can be created and filled with data.

RecyclerView components

- A layout manager:
  - The layout manager handles the organization (layout) of user interface components in a View.
  - 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.
- An adapter.
  - Use an extension of RecyclerView.Adapter to connect your data to the RecyclerView.
  - It prepares the data and how will be displayed in a ViewHolder.
  - When the data changes, the adapter updates the contents of the respective list item view in the RecyclerView

RecyclerView components

- A ViewHolder.
  - Use an extension of RecyclerView.ViewHolder to contain the information for displaying one View item using the item's layout.



Data

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

RecyclerView

- A RecyclerView is a ViewGroup for a scrollable container.
- It is Ideal for long lists of similar items.
- A RecyclerView uses a limited number of View items that are reused 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 View items as can 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 XML layout file so that the adapter can create View items and edit their contents independently from the layout of the Activity.

Layout Manager

- A layout manager positions View items inside a ViewGroup, such as the RecyclerView, and determines when to reuse View items 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 View items in this manner improves performance by avoiding the creation of unnecessary View items 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.

Adapter

- An adapter helps two incompatible interfaces to work together.
- In a RecyclerView, the adapter connects data with View items.
- 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

- The RecyclerView.Adapter implements a ViewHolder, and must override the following callbacks:
- onCreateViewHolder() inflates a View item and returns a new ViewHolder that contains it. This method is called when the RecyclerView needs a new ViewHolder to represent an item.
- onBindViewHolder() sets the contents of a View item at a given position in the RecyclerView. This is called by the RecyclerView, for example, when a new View item scrolls onto the screen.
- getItemCount() returns the total number of items in the data set held by the adapter.

ViewHolder

- A RecyclerView.ViewHolder describes a View item and metadata about its place within the RecyclerView.
- Each ViewHolder holds one set of data.
- The adapter adds data to each ViewHolder for the layout manager to display.
- ViewHolder layout can be defined in an XML resource file

Implementing a RecyclerView

- Implementing a RecyclerView requires the following steps:

1. Add the `RecyclerView` dependency if needed (depending on which template is used for the `Activity`).
2. Add the `RecyclerView` to the `Activity` layout.
3. Create a layout XML file for one `View` item.
4. Extend `RecyclerView.Adapter` and implement the `onCreateViewHolder()` and `onBindViewHolder()` methods.
5. Extend `RecyclerView.ViewHolder` to create a `ViewHolder` for your item layout. You can add click behavior by overriding the `onClick()` method.
6. In the `Activity`, inside the `onCreate()` method, create a `RecyclerView` and initialize it with the adapter and a layout manager.

Adding the dependency

- The RecyclerView library (android.support.v7.widget.RecyclerView) is part of the Support Library.
- Some Activity templates, such as the Basic Activity template, already include the Support Library dependency in the app's build.gradle (Module: app) file.
- If Android Studio doesn't suggest ‹android.support.v7.widget.RecyclerView when entering ‹RecyclerView in the layout editor, you need to add the Support Library dependency for RecyclerView to the dependencies section of the app's build.gradle (Module: app) file:

```
compile 'com.android.support:recyclerview-v7:26.1.0'
```

Adding a RecyclerView to the Activity layout

- Add the RecyclerView to the Activity layout file:
- The only required attributes are the id, the layout_width, and the layout_height.
- For customizing with more attributes, add the attributes to the items, not to this RecyclerView, which is a ViewGroup.

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

Creating the layout for one item

- Create an XML resource file and specify the layout of one item.
- The adapter uses this code to create the ViewHolder

```
<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>
```

Creating the layout for one item

- The TextView 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 View elements.
- An easy way to create a style is to extract the style of a View element that you already created.

1. Right-click (or Control-click) the `TextView`.
2. Choose **Refactor > Extract > Style**.
3. Name your style, leave all other options selected, and select the **Launch 'Use Style Where Possible'** option. Then click **OK**.
4. When prompted, apply the style to the **Whole Project**.

Creating an adapter with a ViewHolder

- 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 context, and your data

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

Creating an adapter with a ViewHolder

- For this adapter, we have to implement three methods:

1. `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);
}
```

Creating an adapter with a ViewHolder

- onBindViewHolder() associates the data with the ViewHolder 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);
}
```

Creating an adapter with a ViewHolder

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

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

Implementing the ViewHolder class

- The ViewHolder class for your item layout is usually defined as an inner class to the adapter.
- Extend RecyclerView.ViewHolder to create the ViewHolder.
- We can add click behavior by overriding the onClick()method.

```
class WordViewHolder extends RecyclerView.ViewHolder {
```

- If you want to add click handling, you need to implement View.onClickListener

```
// Extend the signature of WordViewHolder to implement a click listener.
class WordViewHolder extends RecyclerView.ViewHolder
                                    implements View.OnClickListener {
```

Implementing the ViewHolder class

- In its constructor, the ViewHolder 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 = itemView.findViewById(R.id.word);
    this.mAdapter = adapter;
    itemView.setOnClickListener(this);
}
```

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

Creating the RecyclerView

- Finally, to tie it all together, add to the Activity the following:

1. Declare a `RecyclerView`.

```
private RecyclerView mRecyclerView;
```

2. In the `Activity` `onCreate()` method, get a handle to the `RecyclerView` in the layout:

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

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

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

4. Connect the adapter with the `RecyclerView`.

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

5. Give the `RecyclerView` a default layout manager.

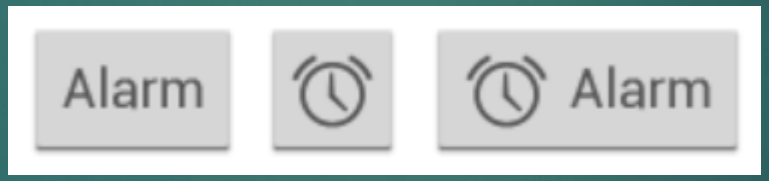
```
mRecyclerView.setLayoutManager(new LinearLayoutManager(this));
```

User Interaction

- The user interface (UI) that appears on a screen of an Android-powered device consists of a hierarchy of objects called views. Every element of the screen is a view.
- The View class represents the basic building block for all UI components
- For an Android app, user interaction typically involves tapping, typing, using gestures, or talking.
- The Android framework provides corresponding user interface (UI) elements such as buttons, clickable images, menus, keyboards, text entry fields, and a microphone.

Buttons

- Button is a user interface that is used to perform some action when clicked or tapped
- It is a very common widget in Android and developers often use it
- We use the Button class to make a button for an Android app
- Generally, Buttons in android will contain a text or an icon or both



- Buttons are sometimes called "push-buttons"

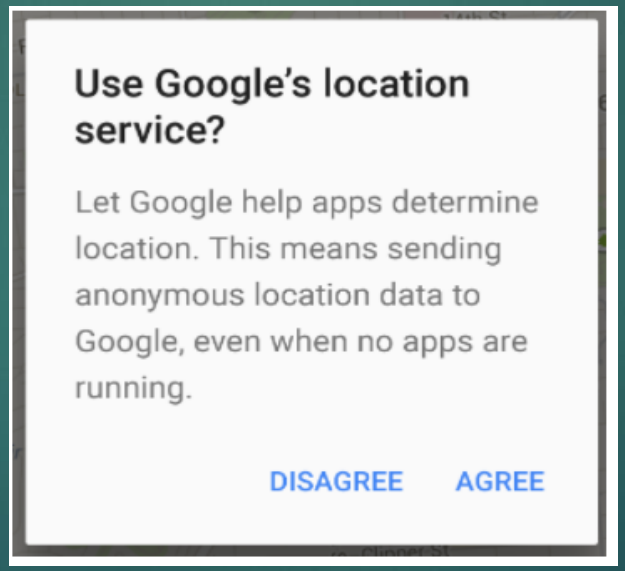
Raised Buttons

- A raised button is an outlined rectangle or rounded rectangle that appears lifted from the screen
- The shading around it indicates that it is possible to tap or click it
- A raised Button is the default style
- Raised button in three states:
- Normal state: A raised Button.

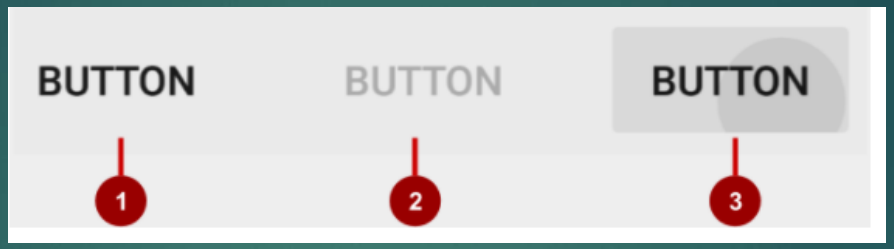
- Disabled state: When disabled, the Button is dimmed out and not active in the app's context. In most cases you would hide an inactive Button, but there may be times when you would want to show it as disabled.
- Pressed state: The pressed state, with a larger background shadow, indicates that the Button is being touched or clicked. When you attach a callback to the Button (such as the android:onClick attribute), the callback is called when the Button is in this state.

Flat buttons/text buttons/borderless buttons

- It is text-only button that looks flat and doesn't have a shadow
- The major benefit of flat buttons is simplicity: a flat button doesn't distract the user from the main content as much as a raised button does

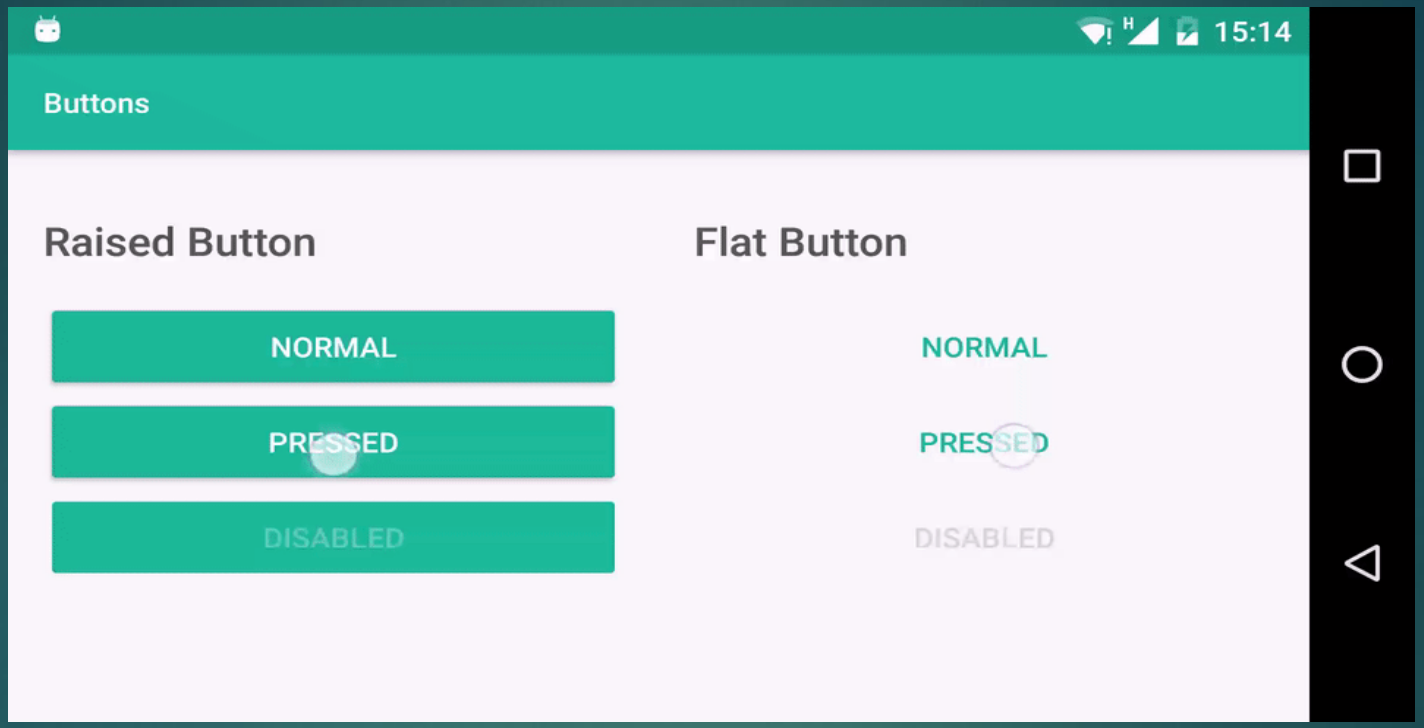


Flat buttons/text buttons/borderless buttons



- Normal state: In its normal state, the button looks just like ordinary text.
- Disabled state: When the text is dimmed out, the button is not active in the app's context.
- Pressed state: A background shadow indicates that the button is being tapped or clicked. When you attach a callback (such as the android:onClick attribute) to the button, the callback is called when the button is in this state.

Flat buttons/text buttons/borderless buttons



Responding to button-click events

- To respond to a user tapping or clicking a button, use the event listener called OnClickListener, which contains one method, onClick()
- To provide functionality when the user clicks, you implement this onClick() method

Adding onClick() to the layout element

- A quick way to set up an OnClickListener for a clickable element in Activity code and assign a callback method is to add the android:onClick attribute to the 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" />
```

- When a user clicks the Button, the Android framework calls the sendMessage() method in the Activity:

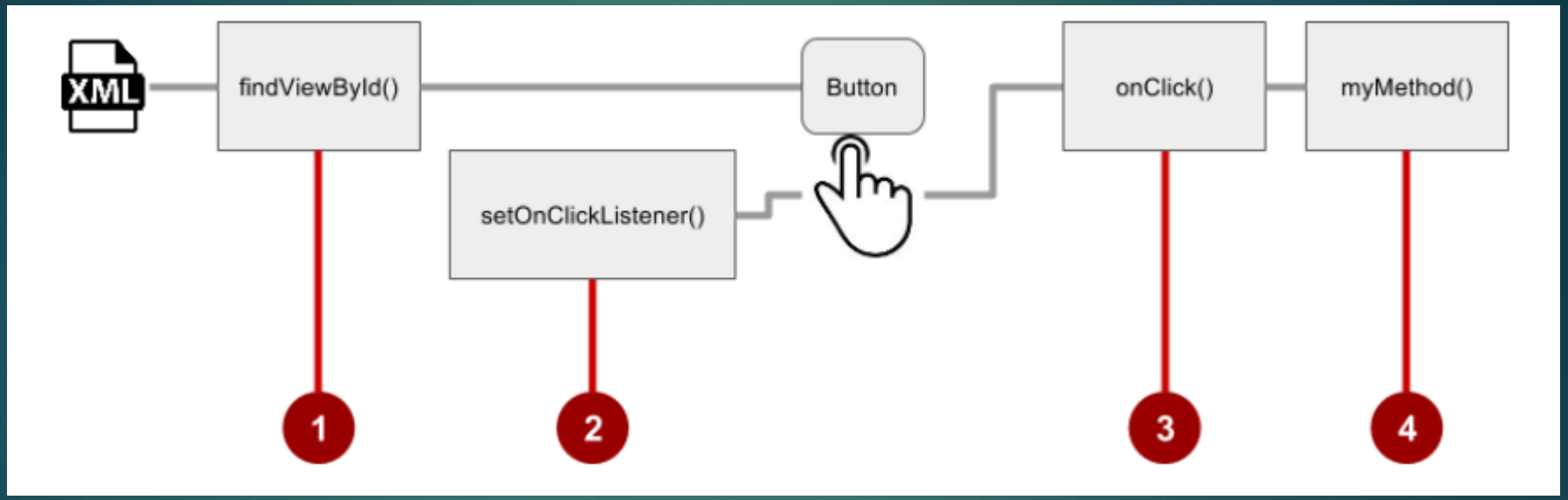
```
public void sendMessage(View view) {
    // Do something in response to button click
}
```

Using the button-listener design pattern

- We can also handle the click event in Java code using the button-listener design pattern
- We need to 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

Using the button-listener design pattern

- Steps to register the listener and use it:



Using the button-listener design pattern

- 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 = findViewById(R.id.button_send);

2. Get a new View.OnClickListener 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() {
    // ... The onClick method goes here.
}
```

Using the button-listener design pattern

3. Override the onClick() method:

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

4. Do something in response to the button click, such as perform an action.

Creating a raised button with an icon and text

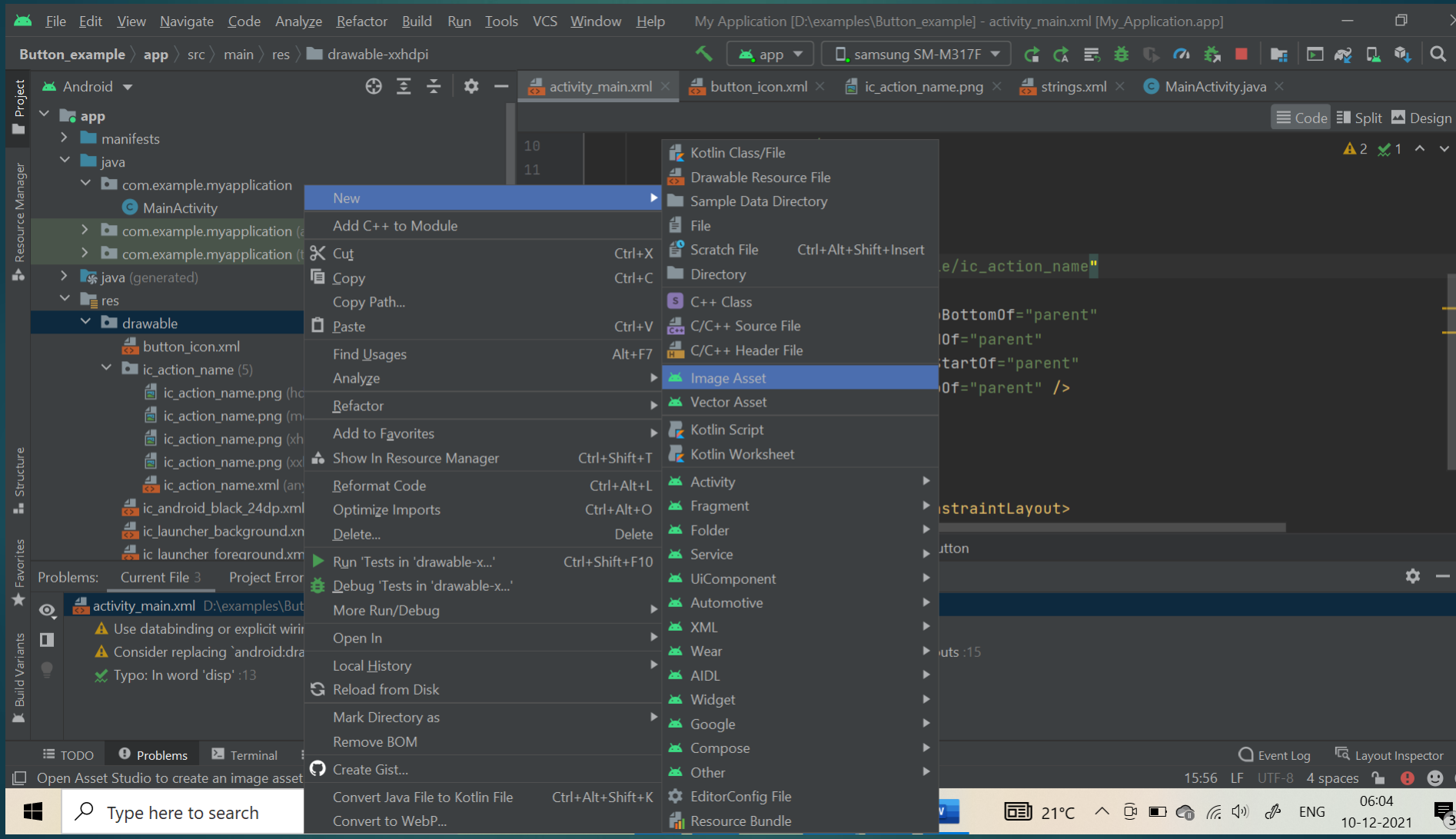
- While a Button usually displays text that tells the user what the action is, a raised Button can also display an icon along with text
- Choosing an icon
- To choose images of a standard icon that are resized for different displays, follow these steps:
- Expand app › res in the Project › Android pane, and right-click (or Command-click) the drawable folder.
- Choose New › Image Asset. The Configure Image Asset dialog appears.
- Choose Action Bar and Tab Icons in the drop-down menu.
- Click the Clipart: image (the Android logo) to select a clip art image as the icon

Creating a raised button with an icon and text

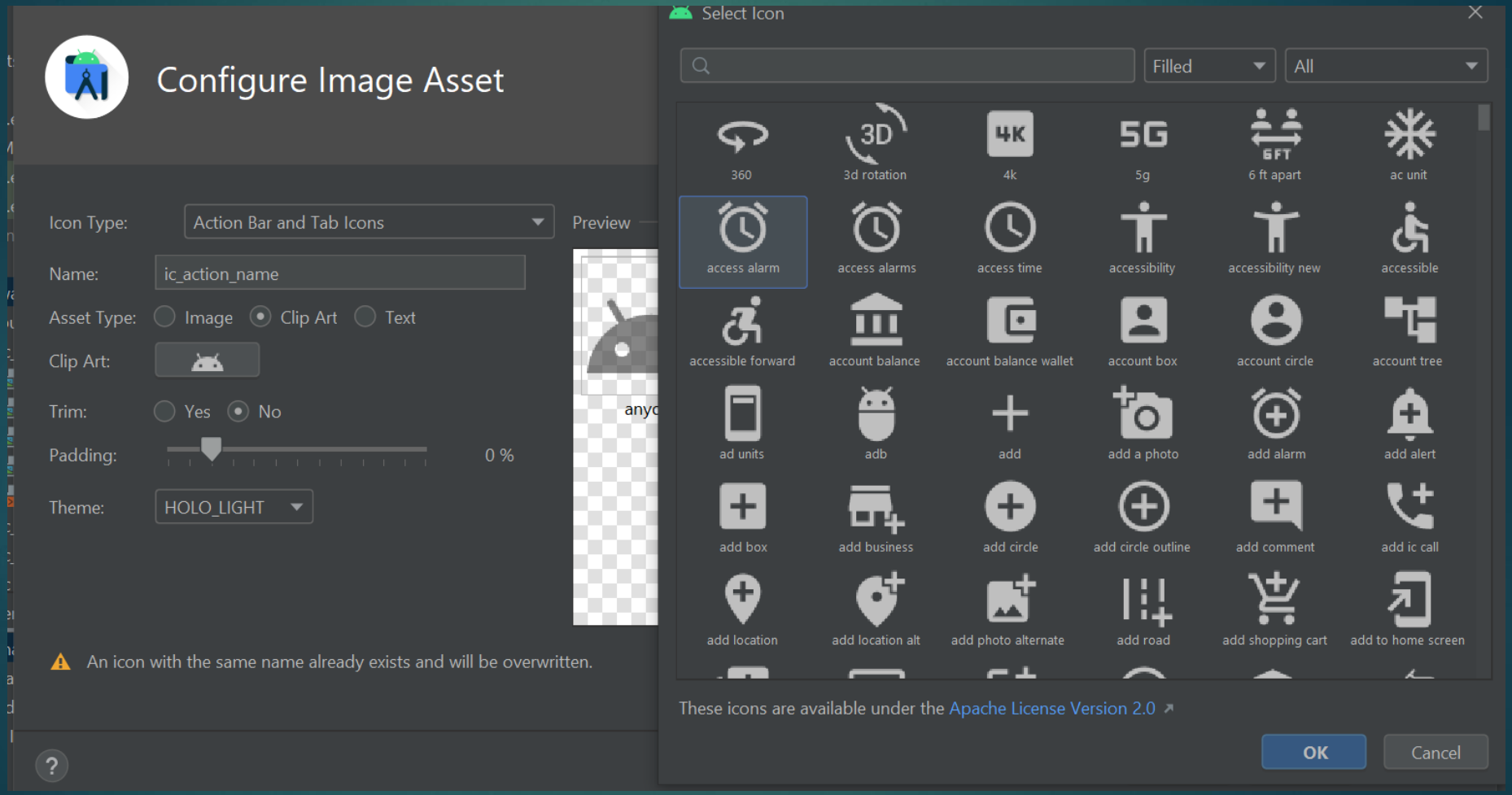
- Optional: Choose HOLO_DARK from the Theme drop-down menu to set the icon to be white against a dark-colored or black background.

- Optional: 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.
- Click Next, and then click Finish in the Confirm Icon Path dialog. The icon name should now appear in the app › res › drawable folder.

Creating a raised button with an icon and text



Creating a raised button with an icon and text



- Vector images of a standard icon are automatically resized for different sizes of device displays. To choose vector images, follow these steps:

- Expand app › res in the Project › Android pane, and right-click (or Command-click) the drawable folder.
- Choose New › Vector Asset for an icon that automatically resizes itself for each display.
- The Vector Asset Studio dialog appears for a vector asset. Click the Material Icon radio button, and then click the Choose button to choose an icon from the Material Design specification.
- Click Next after choosing an icon, and click Finish to finish. The icon name should now appear in the app › res › drawable folder.

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,

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
    android:drawableLeft="@drawable/button_icon"
    <!-- more attributes ... -->
/>
```

Creating a raised button with only an icon

- To create a raised button with just an icon or image (no text), use the ImageButton class, which extends the ImageView class.

```
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/button_icon"
    <!-- more attributes ... -->
/>
```

Adding flat button

- To create a flat button, use the Button class.
- Add a Button to your XML layout, and apply "?android:attr/borderlessButtonStyle" as the style attribute:

```
<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" />
```

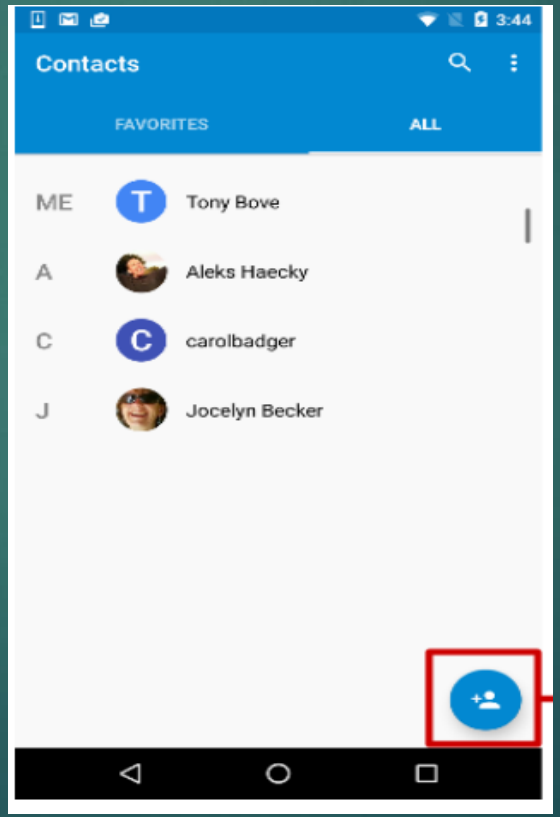
Using clickable images

- We 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 drawable folder of your project.

```
<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/icecream_circle"
    android:onClick="orderIcecream"/>
```

Using a floating action button

- It is a circular button that appears to float above the layout.
- We should use a floating action button only to represent the primary action for a screen



Using a floating action button

- If we start your project or Activity with the Basic Activity template, Android Studio adds a floating action button to the layout file for the Activity.
- To create a floating action button yourself, use the FloatingActionButton class, which extends the ImageButton class

```
<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" />
```

Attributes

- id
- gravity: The gravity attribute is an optional attribute which is used to control the alignment of the text like left, right, center, top, bottom, center_vertical, center_horizontal etc.
- text text="learn android"

  Java: Button button = (Button) findViewById(R.id.simpleButton);

  button.setText("Learn Android")
- textColor: textColor attribute is used to set the text color of a Button. Color value is in the form of "#argb", "#rgb", "#rrggbb", or "#aarrggbb"

  java: Button simpleButton=(Button) findViewById(R.id.simpleButton);

  simpleButton.setTextColor(Color.RED);

Attributes

- textSize: textSize attribute is used to set the size of the text on Button. We can set the text size in sp(scale independent pixel) or dp(density pixel).

  Java: Button simpleButton=(Button)findViewById(R.id.simpleButton);

  simpleButton.setTextSize(25);

textStyle: textStyle attribute is used to set the text style of a Button. The possible text styles are bold, italic and normal. If we need to use two or more styles for a Button then "|" operator is used for that

background: background attribute is used to set the background of a Button. We can set a color or a drawable in the background of a Button

padding: padding attribute is used to set the padding from left, right, top or bottom

drawableBottom: drawableBottom is the drawable to be drawn to the below of the text

TABS AND SWIPES IN ANDROID

SWAROOP SHETTY K

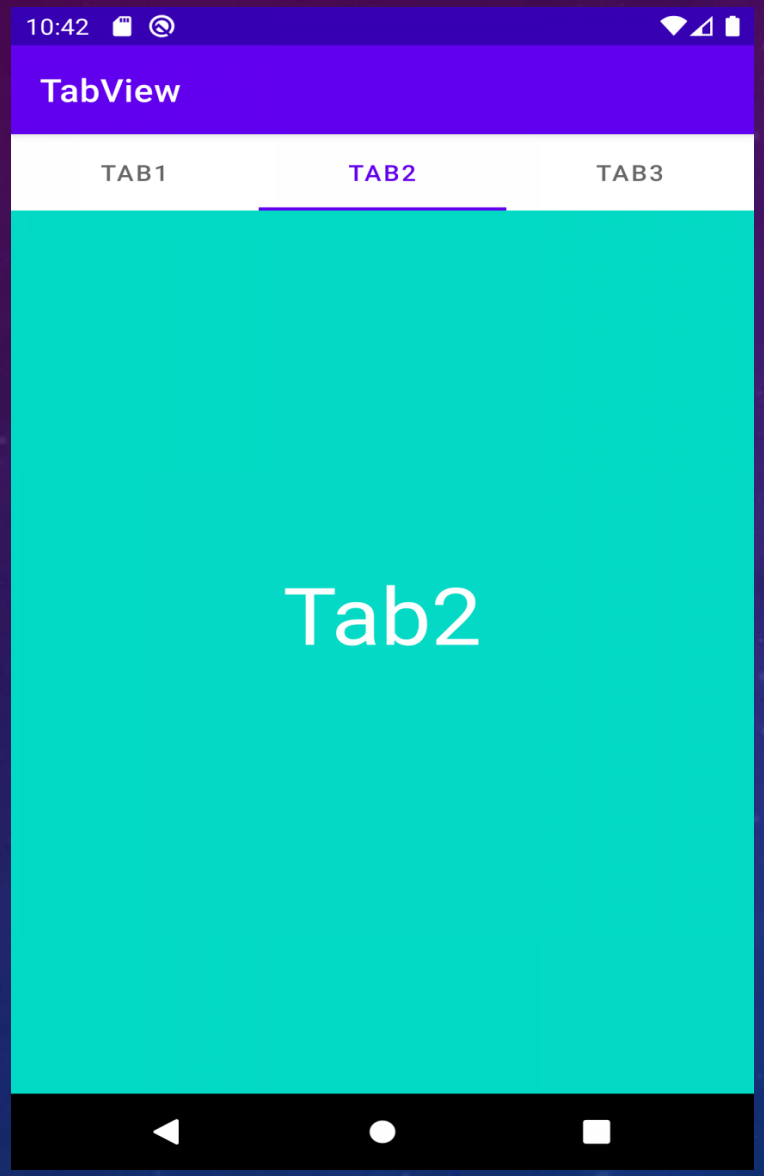
4NM20MC113

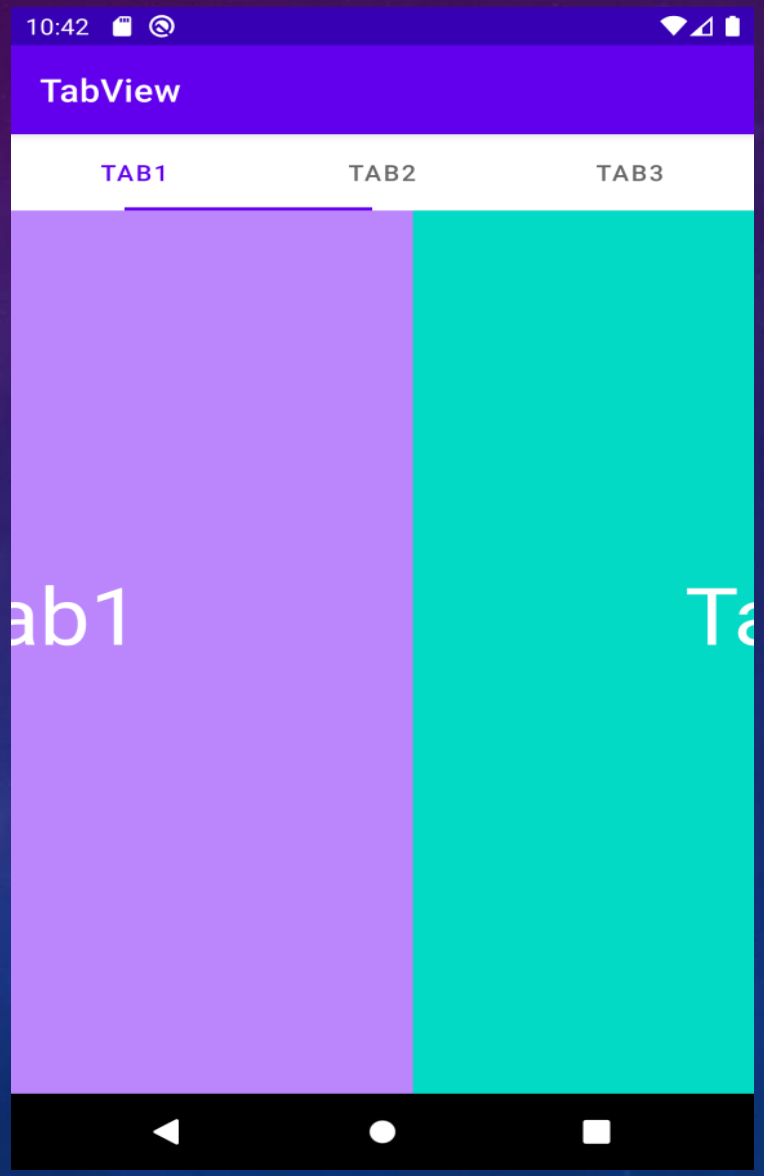
What is tab in android?

In some android apps, Tabs are used, which allows developers to combine multiple tasks (operations) on a single activity. On another side, it provides a different look to that app. It is also possible to provide different feel like left and right swipe by using ViewPager.

How to create swipeable tabs layout in Android?

In android, we can create swipeable tabs layout using Swipe Views to switch between the tabs in android application. The Swipe Views in android provides a navigation between the sibling screens such as tabs with a horizontal finger gesture, sometimes it is called as horizontal paging.


10:42
TabView
TAB1
TAB2
TAB3
Tab2


10:42
TabView
TAB1
TAB2
TAB3
ab1
Ta

TabLayout is used to implement horizontal tabs. TabLayout is released by Android after the deprecation of *ActionBar.TabListener (API level 21).*

TabLayout is introduced in design support library to implement tabs.

Tabs are created using *newTab()* method of TabLayout class. The title and icon of Tabs are set through *setText(int)* and *setIcon(int)* methods of TabListener interface respectively. Tabs of layout are attached over TabLayout using the method addTab(Tab) method.

This example demonstrates how do I create a Tab Layout in android app.

Step 1 − Create a new project in Android Studio, go to File ⇒ New Project and fill all required details to create a new project.

Step 2 – Add the following dependency to create a tab layout −

implementation 'com.android.support:design:28.0.0'

Step 3 − Add the following code to res/layout/activity_main.xml.

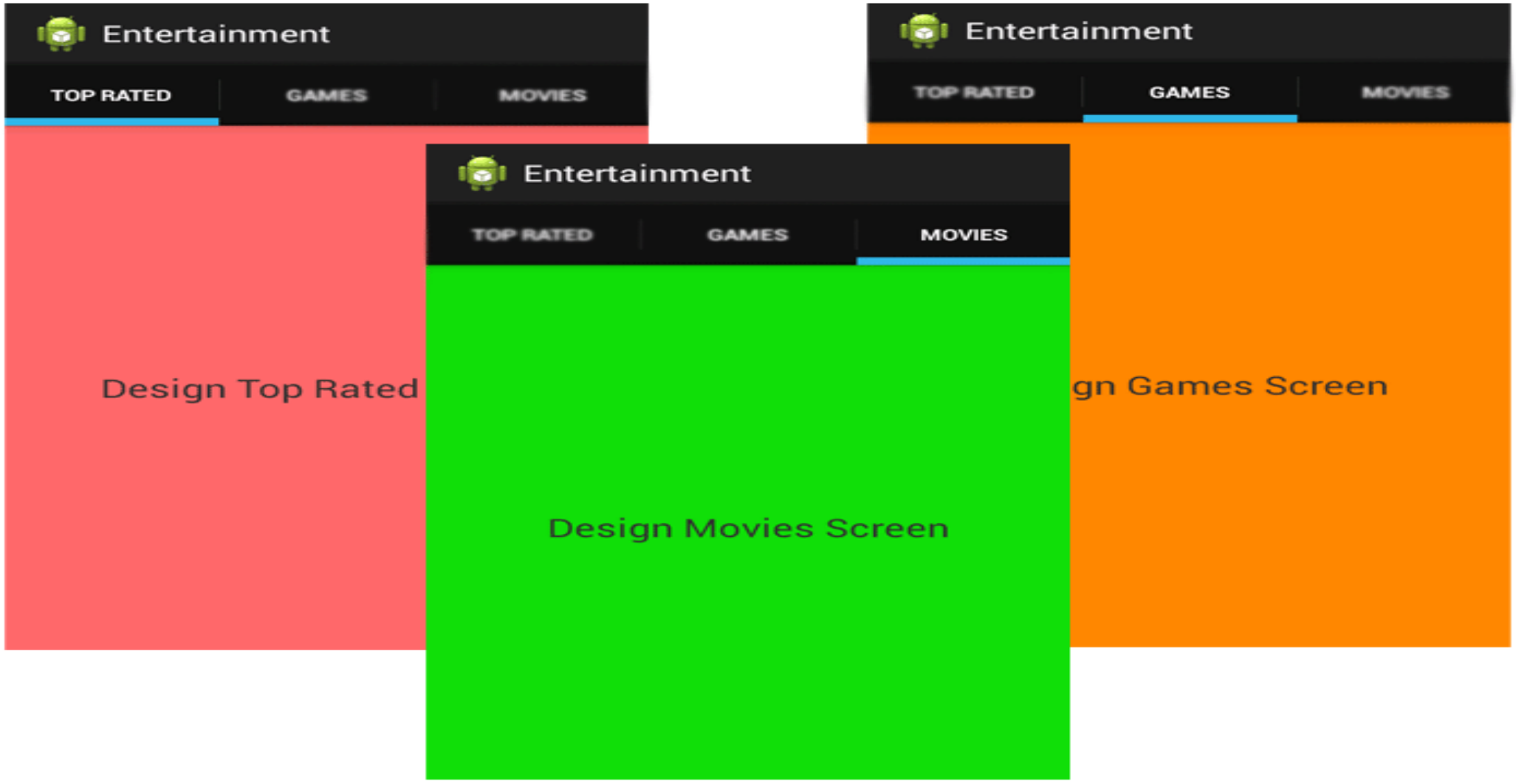
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".MainActivity">
```

```
<android.support.design.widget.TabLayout
    android:id="@+id/tabLayout"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="#1db995">
  </android.support.design.widget.TabLayout>
  <android.support.v4.view.ViewPager
    android:id="@+id/viewPager"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@id/tabLayout"
    android:layout_centerInParent="true"
    android:layout_marginTop="100dp"
    tools:layout_editor_absoluteX="8dp" />
</RelativeLayout>
```

Step 4 − Add the following code to src/MainActivity.java

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.design.widget.TabLayout;
import android.support.v4.view.ViewPager;
public class MainActivity extends AppCompatActivity {
   TabLayout tabLayout;
   ViewPager viewPager;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      tabLayout = findViewById(R.id.tabLayout);
      viewPager = findViewById(R.id.viewPager);
      tabLayout.addTab(tabLayout.newTab().setText("Football"));
      tabLayout.addTab(tabLayout.newTab().setText("Cricket"));
      tabLayout.addTab(tabLayout.newTab().setText("NBA"));
      tabLayout.setTabGravity(TabLayout.GRAVITY_FILL);
```

```
final MyAdapter adapter = new MyAdapter(this,getSupportFragmentManager(),
        tabLayout.getTabCount());
        viewPager.setAdapter(adapter);
        viewPager.addOnPageChangeListener(new TabLayout.TabLayoutOnPageChangeListener(tabLayout));
        tabLayout.addOnTabSelectedListener(new TabLayout.OnTabSelectedListener() {
            @Override
            public void onTabSelected(TabLayout.Tab tab) {
                viewPager.setCurrentItem(tab.getPosition());
            }
            @Override
            public void onTabUnselected(TabLayout.Tab tab) {
            }
            @Override
            public void onTabReselected(TabLayout.Tab tab) {
            }
        });
    }
}
```


Entertainment
TOP RATED
GAMES
MOVIES
Design Top Rated
Entertainment
TOP RATED
GAMES
MOVIES
Design Movies Screen
Entertainment
TOP RATED
GAMES
MOVIES
gn Games Screen

THANK YOU