User Experience

In this Lecture, you will learn:

- Android UI Development
- Supporting different pixel densities
- UI Views

User Interface

The user interface (UI) on an Android-powered device consists of a hierarchy of objects called views. View is the base class for classes that provide interactive UI components, such as Button elements. You can make any View that can be tapped or clicked.

Views VS ViewGroups:

View class represents the basic building block for user interface components. A View occupies a rectangular area on the screen and is responsible for drawing and event handling. View is the base class for widgets, which are used to create interactive UI components (buttons, text fields, etc.).

Examples:

TextView:

EditText:

Spinner

...

ViewGroup is a special view that contains other views (called children.) The view group is the base class for layouts and views containers. This class also defines the ViewGroup.LayoutParams class which serves as the base class for layouts parameters.

Examples:

LinearLayout:

Relative Layout:

...

Supporting different pixel densities

Android devices come in different screen sizes and different pixel sizes. One device has 160 pixels per square inch, while another device fits 480 pixels in the same space.

Your app needs to support different pixel densities by using resolution-independent units of measurements and providing alternative bitmap resources for each pixel density.

Use density-independent pixels:

The pitfall to avoid is using pixels to define distances or sizes. Defining dimensions with pixels is a problem because different screens have different pixel densities, so the same number of pixels may correspond to different physical sizes on different devices.

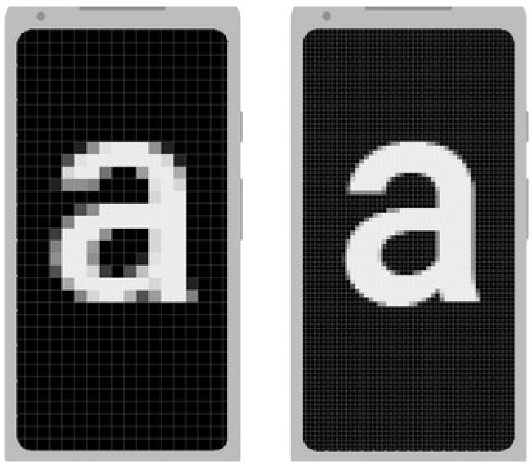


Figure 1. Two screens of the same size may have a different number of pixels

To preserve the visible size of your UI on screens with different densities, design your UI using density-independent pixels (dp) as unit of measurement. One dp is a virtual pixel unit, roughly equal to one pixel on a medium-density screen (160dpi; the "baseline" density). Android translates this value to an appropriate number of real pixels for each other density.

For example, consider the two devices in figure 1. If you define a view to be "100px" wide, it will appear much larger on the device on the left. So you must instead use "100dp" to ensure it appears the same size on both screens.

When defining text sizes, use scalable pixels (sp) as units. The sp unit is the same size as dp, by default, but it resizes based on the user's preferred text size.

While specify spacing between two views, use dp:

```
<Button android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="@string/clickme"
   android:layout_marginTop="20dp" />
```

When specifying text size, always use sp:

```
<TextView android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:textSize="20sp" />
```

Views:
A view like button can be made clickable by adding the android:onClick attribute in the XML layout. Or you can make an image act like a button by adding android:onClick to the ImageView.

Add images/icons to project:

→Copy the image files into your project's drawable folder. Find the drawable folder in a project by using this path: project_name > app > src > main > res > drawable.
→Open your project.
→Open your layout xml file, and click the Design tab.
→Drag an ImageView to the layout, and choose the added image for it.

ImageView:
An ImageView comes with different options to support different scale types. Scale type options are used for scaling the bounds of an image to the bounds of the imageview. scaleTypes configuration properties include center, center_crop, fit_xy, fitStart.

The following XML layout code uses an ImageView:

```
<ImageView
android:id="@+id/simpleImageView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:src="@drawable/donut_circle" />
```

Make sure to put an image named *donut_circle*.png in drawable folder of the project.

To add an image to the project follow the steps given above.

Attributes of ImageView:

Some important attributes that configure an ImageView in your xml file include:

id: id is an attribute used to uniquely identify any element in android. Below is the example code in which we set the id of an ImageView.

```
<ImageView
android:id="@+id/simpleImageView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
/>
```

src: src attribute sets a image source file.

Below is the example code in which we set the source of an ImageView donut_circle which is saved in drawable folder.

```
<ImageView
android:id="@+id/simpleImageView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:src="@drawable/donut_circle" />
```

In Java:
We can also set the source image at run time programmatically in java class. For that we use setImageResource() method as shown in below example code.

```
/*Add in Oncreate() funtion after setContentView()*/
ImageView simpleImageView=(ImageView)
findViewById(R.id.simpleImageView);
simpleImageView.setImageResource(R.drawable.donut_circle);
```

background: background attribute sets the background of an ImageView. We can set a color or a drawable in the background of an ImageView.

Example below sets black color in the background and an image in the src attribute of ImageView.

```
<ImageView
   android:id="@+id/simpleImageView"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:src="@drawable/donut_circle"
   android:background="#000"/>
```

Java:
We can also set the background at run time in java class. In below example code we set the black color in the background of a image view.

```
ImageView simpleImageView = findViewById(R.id.simpleImageView);
simpleImageView.setBackgroundColor(Color.BLACK);
```

padding: padding attribute sets the padding from left, right, top or bottom of an Imageview.

paddingRight: set the padding from the right side of the image view.
paddingLeft: set the padding from the left side of the image view.
paddingTop: set the padding from the top side of the image view.
paddingBottom: set the padding from the bottom side of the image view.
padding: set the padding from the all side’s of the image view.

Example code below sets 30dp padding from all sides of an ImageView.

```
<ImageView
   android:id="@+id/simpleImageView"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:background="#000"
   android:src="@drawable/donut_circle"
   android:padding="30dp"/>
```

scaleType: scaleType attribute controls how the image should be re-sized or moved to match the size of the ImageView. The value for scaleType attribute can be fit_xy, center_crop, fitStart etc.

Below is the example of scale type fit_xy.

```
<ImageView
android:id="@+id/simpleImageView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:src="@drawable/donut_circle"
android:scaleType="fitXY"/>
```

The value for scaleType “fitStart” is used to fit the image in the start of the ImageView as shown below:

```
<ImageView
    android:id="@+id/simpleImageView"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:src="@drawable/donut_circle"
    android:scaleType="fitStart"/>
```

Add onClick methods for images:

To make a View clickable so that users can tap (click) it, add the android:onClick attribute in the XML layout and specify the click handler in Java code.

onClick:

```
<ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="10dp"
        android:id="@+id/donut"
        android:src="@drawable/donut_circle"
        android:onClick="showDonutOrder"/>
```

click handler:

```
public void showDonutOrder(View view) {
    Toast.makeText(getApplicationContext(), “You ordered a donut”,
                        Toast.LENGTH_SHORT).show();
}
```

Full Program:

activity_main.xml

```
<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">

    <ImageView
        android:id="@+id/simpleImageViewDonut"
        android:layout_width="fill_parent"
        android:layout_height="200dp"
        android:scaleType="fitXY"
        android:src="@drawable/donut_circle" />

    <ImageView
    android:id="@+id/simpleImageIcecream"
    android:layout_width="fill_parent"
    android:layout_height="200dp"
    android:layout_below="@+id/simpleImageViewDonut"
    android:layout_marginTop="10dp"
    android:scaleType="fitXY"
    android:src="@drawable/icecream_circle" />

</RelativeLayout>
```

Code to initiate the ImageView and implement click event on them.
MainActivity.java:

```
//import statements are omitted
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ImageView simpleImageViewDonut = (ImageView)
findViewById(R.id.simpleImageViewDonut);
        ImageView simpleImageViewIcecream = (ImageView)
findViewById(R.id.simpleImageViewIcecream);

        simpleImageViewDonut.setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Toast.makeText(getApplicationContext(),
"simpleImageViewDonut", Toast.LENGTH_LONG).show();
            }
        });
        simpleImageViewIcecream.setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Toast.makeText(getApplicationContext(),
"simpleImageViewIcecream", Toast.LENGTH_LONG).show();
            }
        });
    }
}
```

Input Controls:

To enter text or numbers, use an EditText. Some input controls are EditText attributes that define the type of keyboard that appears. For example, choose phone for the android:inputType attribute to show a numeric keypad instead of an alphanumeric keyboard.

A RadioButton enables user to select only one item from a set of items.

EditText:

Editable text field expects a certain type of text input, such as plain text, an email address, a phone number, or a password. Specify the input type for each text field in your app so that the system displays the appropriate input method, such as an on-screen keyboard for plain text, or a numeric keypad for entering a phone number.

EditText attributes for entering a name:

android:hint	*"Enter Name"*
android:inputType	*"textPersonName"*

Multiple-line EditText:

Add an EditText element. To use the visual layout editor, drag a Multiline Text element from the Palette pane. Check the XML code by clicking the Text tab. The following attributes should be set for the new EditText:

android:inputType	*"textMultiLine"*

Phone numbers:

android:inputType	*"phone"*

Combine input types in one EditText

You can combine inputType attribute values that don't conflict with each other. For example, you can combine the textMultiLine and textCapSentences attribute values for multiple lines of text in which each sentence starts with a capital letter.

android:inputType	*"textCapSentences|textMultiLine"*

Floating Action Button

A floating action button is a round button. The following code shows how the FloatingActionButton should appear in your layout file:

```
<android.support.design.widget.FloatingActionButton
        android:id="@+id/floatingActionButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentEnd="true"
        android:layout_alignParentBottom="true"
        android:layout_marginBottom="286dp"
        android:clickable="true"
        app:srcCompat="@android:drawable/btn_dialog" />
```

By default, FAB is colored by the colorAccent attribute. You can configure other FAB properties using either XML attributes or corresponding methods, such as:

The size of the FAB, using the app:fabSize attribute or the setSize() method. The FAB icon, using the android:src attribute or the setImageDrawable() method.

Handling button tap

The View.OnClickListener handles tap events. For example, the following code displays a Snackbar when user taps the FAB:

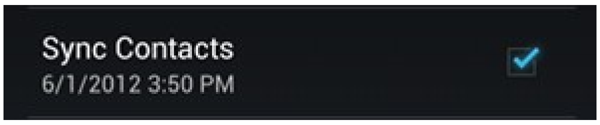
```
FloatingActionButton fab = findViewById(R.id.floatingActionButton);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Snackbar.make(view, "Here's a Snackbar",
Snackbar.LENGTH_SHORT)
                        .setAction("Action", null).show();
            }
        });
```

Snackbar:

Snackbar in android is a new widget introduced with the Material Design library as an alternative of Toast. A Snackbar can be only showed in the bottom of the screen. Snackbar may have action button. Snackbar can be swiped off before the time limit.

Checkboxes

Checkboxes allow the user to select one or more options from a set.


Sync Contacts
6/1/2012 3:50 PM

To create each checkbox option, create a CheckBox in your layout. Since a set of checkbox options allows the user to select multiple items, each checkbox is managed separately and register a click listener for each one.

Click Events

When user selects a checkbox, the CheckBox object gets onclick event. To implement this click event handler, add the android:onClick attribute to the CheckBox element in your XML layout. The value for this attribute is the name of the method you want to call in response to a click event. The Activity hosting the layout must implement the method.

Example:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <CheckBox android:id="@+id/checkBoxApple"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/meat"
        android:onClick="onCheckboxClicked"/>
    <CheckBox android:id="@+id/checkBoxOrange"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/cheese"
        android:onClick="onCheckboxClicked"/>
</LinearLayout>
```

Following method handles the click event for both checkboxes:

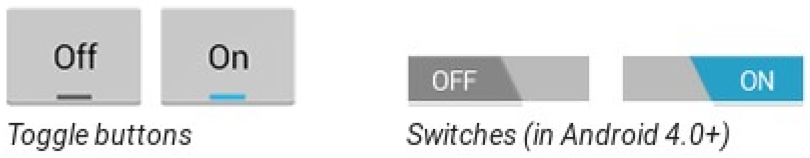
```
public void onCheckboxClicked(View view) {
    boolean isChecked = ((CheckBox) view).isChecked();

    // Check which checkbox was clicked
    switch (checkId){
            case R.id.checkBoxApple:
                if(isChecked ==true ) {
Toast.makeText(this, "Apple checked", Toast.LENGTH_SHORT).show();
                }else{
Toast.makeText(this, "Apple unchecked", Toast.LENGTH_SHORT).show();
                }
                return;

            case R.id.checkBoxOrange:
Toast.makeText(this, "Orange checked", Toast.LENGTH_SHORT).show();
                return;
        }
}
```

Toggle Buttons and Switches:

A toggle button allows to change a setting between two states. Add a basic toggle button to your layout with the ToggleButton object. Android 4.0 (API level 14) introduces new kind of toggle button called a Switch that provides a slider control.



Key classes:

- ToggleButton
- Switch
- SwitchCompat
- CompoundButton

Responding to Button Presses

Create CompoundButton.OnCheckedChangeListener object and assign it to the button by calling setOnCheckedChangeListener(). Example:

```
//Switch toggle = (Switch) findViewById(R.id.switch);
ToggleButton toggle = (ToggleButton) findViewById(R.id.togglebutton);
toggle.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButton buttonView, boolean
isChecked) {
        if (isChecked) {
            // The toggle is enabled
        } else {
            // The toggle is disabled
        }
    }
});
```

<u>**Radio buttons**</u>

Since radio button selections are mutually exclusive, group them together inside a RadioGroup. With grouping, Android ensures that only one radio button can be selected at a time. To add radio buttons in an Activity, create RadioButton elements in the xml layout file.
Example:

```
<RadioGroup
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="24dp"
        android:orientation="vertical">

        <RadioButton
            android:id="@+id/sameday"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="onRadioButtonClicked"
            android:text="Same day messenger service" />
        <RadioButton
            android:id="@+id/nextday"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="onRadioButtonClicked"
            android:text="Next day ground delivery" />
        <RadioButton
            android:id="@+id/pickup"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="onRadioButtonClicked"
            android:text="Pick up" />
</RadioGroup>
```

The "onRadioButtonClicked" value for android:onClick attribute is the call back method for each radio button click.

Radio button click handler

The android:onClick attribute for each radio button element specifies the onRadioButtonClicked() method to handle the click event. Add a onRadioButtonClicked() method in the Activity class.

To add the onClick handler from XML:

Open the xml layout file find one of the onRadioButtonClicked values for the android:onClick attribute that is underlined in red.

Click the onRadioButtonClicked value, and then click the red bulb warning icon in the left margin.

Choose Create onRadioButtonClicked(View) in the Activity in the red bulb's menu. Android Studio creates the onRadioButtonClicked(View view) method in the corresponding Activity as:

```
public void onRadioButtonClicked(View view) {
}
```

```
public void onRadioButtonClicked(View view) {
    // Is the button now checked?
    boolean checked = ((RadioButton) view).isChecked();
    // Check which radio button was clicked.
    switch (view.getId()) {
      case R.id.sameday:
        if (checked)
          // Same day service
          Toast.makeText(this, "Same day service clicked",
               Toast.LENGTH_SHORT).show();
        break;
      case R.id.nextday:
        if (checked)
          // Next day delivery
          Toast.makeText(this, "Next day delivery clicked",
               Toast.LENGTH_SHORT).show();
        break;
      case R.id.pickup:
        if (checked)
```

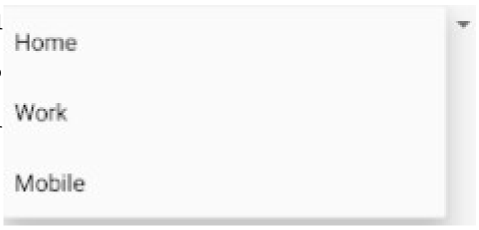
```
                // Pick up
                Toast.makeText(this, “Pick up clicked”,
                    Toast.LENGTH_SHORT).show();
            break;
        default:
            break;
    }
}
```

Run the code and click the radio buttons.

Task:
Make one of the radio button selected by default.

Spinner

A Spinner allows selecting 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 there are only two or three choices, use radio buttons depending on available space in layout. But with more choices, a Spinner works better, scrolls as needed to display items, and takes up little room in your layout.



You can add a spinner to your layout with the Spinner object. You should usually do so in your XML layout with a <Spinner> element.

```
<Spinner
        android:id="@+id/spinner"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentStart="true"
        android:layout_alignParentBottom="true"
        android:layout_marginStart="33dp"
        android:layout_marginBottom="156dp" />
```

To populate the spinner with a list of choices, specify a SpinnerAdapter in your Activity.

Key classes:

Spinner
SpinnerAdapter
AdapterView.OnItemSelectedListener

Populate the Spinner with Options

The options for spinner can come from any source, but must be provided through an SpinnerAdapter. Use ArrayAdapter if the choices are available in an array, or use a CursorAdapter if the choices are available from a database query.

Example: Choice for spinner in string array defined in string resource file:

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

For an array use the following code in your Activity or Fragment to supply the spinner with the array using an instance of ArrayAdapter:

```
Spinner spinner = (Spinner) findViewById(R.id.spinner);
// ArrayAdapter using the string array and a 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_dro
pdown_item);
// Apply the adapter to the spinner
spinner.setAdapter(adapter);
```

The createFromResource() method creates an ArrayAdapter from the string array. The third argument for this method is a 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 need to define your own layout for the spinner's appearance.
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).
Finally call setAdapter() to apply the adapter to your Spinner.

User Selections
When user selects an item from the drop-down, the Spinner object receives an on-item-selected event. To define selection event handler, implement

AdapterView.OnItemSelectedListener interface and the corresponding onItemSelected() callback method. For example, here's an implementation of the interface in an Activity:

```
public class MainActivity extends AppCompatActivity  implements
AdapterView.OnItemSelectedListener {
   ...
    public void onItemSelected(AdapterView<?> adapterView, View view,
                              int pos, long id) {
        String spinnerLabel =
adapterView.getItemAtPosition(pos).toString();
        Toast.makeText(getApplicationContext(), spinnerLabel,
                Toast.LENGTH_SHORT).show();
    }

    public void onNothingSelected(AdapterView<?> parent) {

    }
}
```

The AdapterView.OnItemSelectedListener requires the onItemSelected() and onNothingSelected() callback methods. Specify the interface implementation by calling setOnItemSelectedListener():

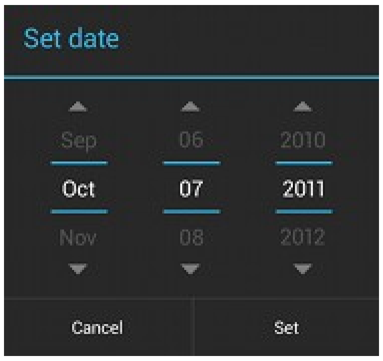
```
Spinner spinner = (Spinner) findViewById(R.id.spinner);
spinner.setOnItemSelectedListener(this);
```

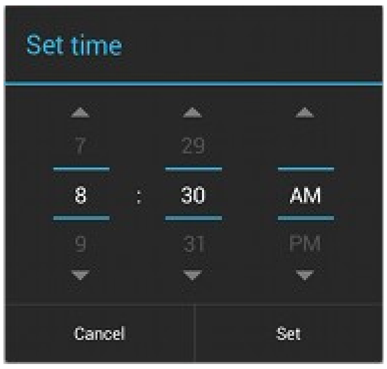
When you implement the AdapterView.OnItemSelectedListener interface in your Activity or Fragment, this object is passed as the interface instance.

Task:
Create a phone number input field and then a spinner in the same line with options: Mobile, Home, Work and Other.

Pickers

Picker controls allow user to pick a time or a date as dialogs. Each picker provides controls for selecting each part of the time (hour, minute, AM/PM) or date (month, day, year). Pickers ensure that users pick a time or date that is valid, formatted correctly, and adjusted to the user's locale.





Time Picker:

To display a TimePickerDialog, define a fragment class that extends DialogFragment and return a TimePickerDialog from the fragment's onCreateDialog() method.

Define the onCreateDialog() method to return an instance of TimePickerDialog. Implement the TimePickerDialog.OnTimeSetListener interface to receive a callback when the user sets the time.

```
public class TimePickerFragment extends DialogFragment
        implements TimePickerDialog.OnTimeSetListener {
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Use the current time as the default values for the picker
        final Calendar calendar = Calendar.getInstance();
        int hour = calendar.get(Calendar.HOUR_OF_DAY);
        int minute = calendar.get(Calendar.MINUTE);

        // Create a new instance of TimePickerDialog and return it
        return new TimePickerDialog(getActivity(), this, hour, minute,
                DateFormat.is24HourFormat(getActivity()));
    }
    public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
    }
}
```

Showing the time picker
Once a DialogFragment is defined, display the time picker by creating an instance of the DialogFragment and calling show() method.
Example:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Time Picker"
    android:onClick="showTimePickerDialog" />
```

When the user clicks this button, the system calls the following method:

```
public void showTimePickerDialog(View v) {
    DialogFragment newFragment = new TimePickerFragment();
    newFragment.show(getSupportFragmentManager(), "timePicker");
}
```

Date Picker:
To display a DatePickerDialog, define a fragment class that extends DialogFragment and return a DatePickerDialog from the fragment's onCreateDialog() method.
Define the onCreateDialog() method to return an instance of DatePickerDialog. Implement the DatePickerDialog.OnDateSetListener interface to receive a callback when the user sets the date.

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

```
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Use the current date as the default date in the picker
        final Calendar calendar = Calendar.getInstance();
        int year = calendar.get(Calendar.YEAR);
        int month = calendar.get(Calendar.MONTH);
        int day = calendar.get(Calendar.DAY_OF_MONTH);

        // Create a new instance of DatePickerDialog and return it
        return new DatePickerDialog(getActivity(), this, year, month, day);
    }

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

Showing the date picker:
Display the date picker by creating an instance of the DialogFragment and calling show() method.
Example:

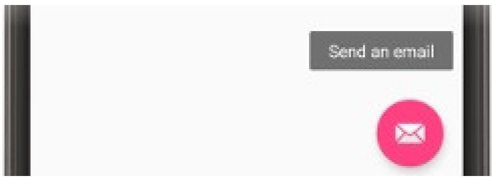
```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Date Picker"
    android:onClick="showDatePickerDialog" />

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

Tooltip:

A tooltip is a small descriptive message that appears near a view when user long presses the view which also resembles the mouse hover event. This is useful when icon is used to represent an action.

You can use tooltips to display a descriptive message, as shown in the figure.



A tooltip text is set in a View by calling setTooltipText() method. Or set the tooltipText property of a view using the corresponding XML attribute as shown in the following example:

```
<android.support.design.widget.FloatingActionButton
    android:id="@+id/fab"
    android:tooltipText="Send an email" />
```

To specify tooltip text in code, use the setTooltipText(CharSequence) method as:

```
FloatingActionButton fab = findViewById(R.id.fab);
fab.setTooltipText("Send an email");
```