User Input Controls

Contents

- User Interaction
- Focus
- Text input and keyboards
- Radio Buttons and Checkboxes
- Making Choices
    - dialogs, spinners and pickers
- Recognizing gestures

User Interaction

Users expect to interact with apps

- Clicking, pressing, talking, typing, and listening

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

- Navigating between activities

User interaction design

Important to be obvious, easy, and consistent:

● Think about how users will use your app

● Minimize steps

● Use UI elements that are easy to access, understand, use

● Follow Android best practices

● Meet user's expectations

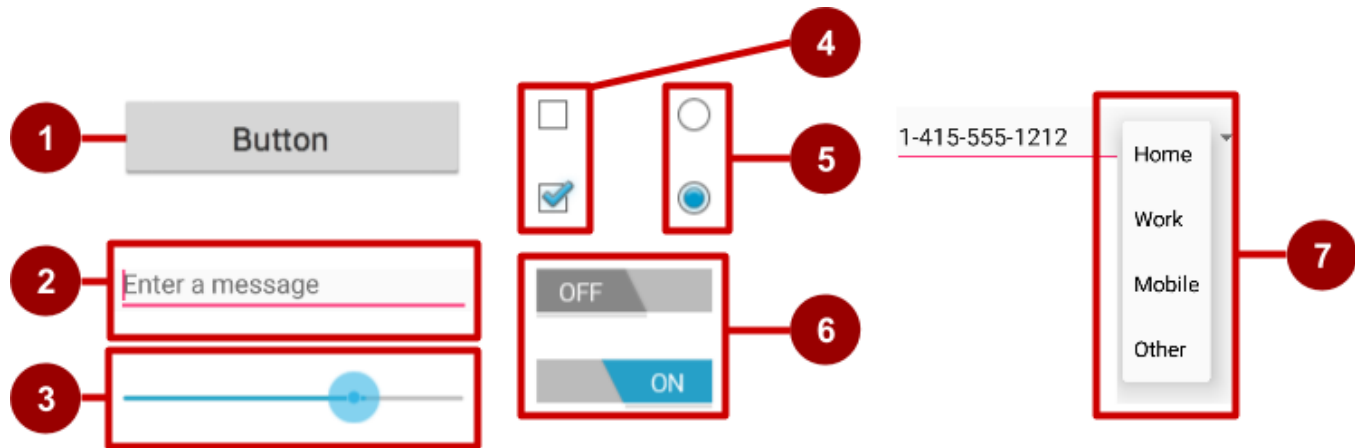# Input Controls

Ways to get input from the user

- Free form
  - Text and voice input

- Actions
  - Buttons
  - Contextual menus
  - Gestures
  - Dialogs

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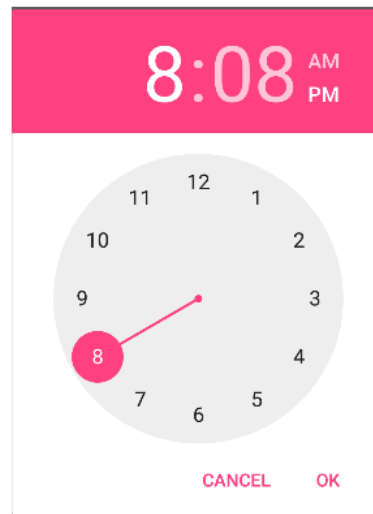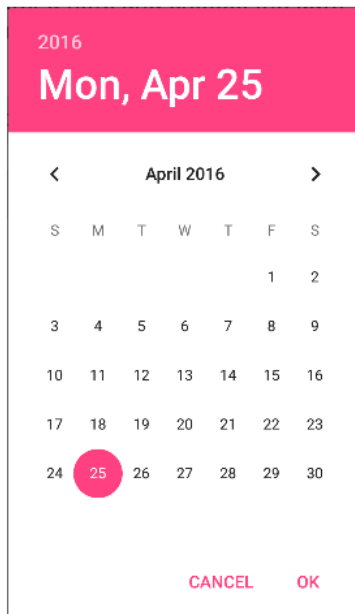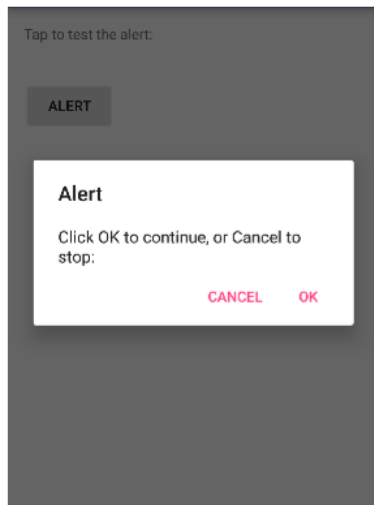 - Toggle buttons
  - Spinners

User Input Controls

Examples of user input controls

1. Button

2. Text field

3. Seek bar

4. Checkboxes

5. Radio buttons

6. Toggle

7. Spinner

Alert dialog, date picker, time picker

View is base class for input controls

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

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

- View provides basic interaction through `android:onClick`

Focus

Focus

- The view that receives user input has "Focus"
- Only one view can have focus
- Focus makes it unambiguous which view gets the input
- Focus is assigned by
  - User tapping a view
  - App guiding the user from one text input control to the next using the Return, Tab, or arrow keys
  - Calling `requestFocus()` on any view that is focusable

Clickable versus focusable

Clickable—View can respond to being clicked or tapped

Focusable—View can gain focus to accept input

Input controls such as keyboards send input to the view that has focus

Which View gets focus next?

- Topmost view under the touch

- After user submits input, focus moves to nearest neighbor—priority is left to right, top to bottom

- Focus can change when user interacts with a directional control

Guiding users

- Visually indicate which view has focus so users knows where their input goes

- Visually indicate which views can have focus helps users navigate through flow

- Predictable and logical—no surprises!

Guiding focus

- Arrange input controls in a layout from left to right and top to bottom in the order you want focus assigned
- Place input controls inside a view group in your layout
- Specify ordering in XML

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

android:focusable="true"

android:nextFocusDown="@+id/bottom"
```

User Input
Controls

Set focus explicitly

Use methods of the View class to set focus

- setFocusable() sets whether a view can have focus

- requestFocus() gives focus to a specific view

- setOnFocusChangeListener() sets listener for when view gains or loses focus

- onFocusChanged() called when focus on a view changes

Find the view with focus

- [Activity.getCurrentFocus()](#)

- [ViewGroup.getFocusedChild()](#)

Text Input

EditText

- [EditText](#) class

- Multiple lines of input

- Characters, numbers, and symbols

- Spelling correction

- Tapping the Return (Enter) key starts a new line

- Customizable

"Action" key

Android Developer Fundamentals

User Input Controls
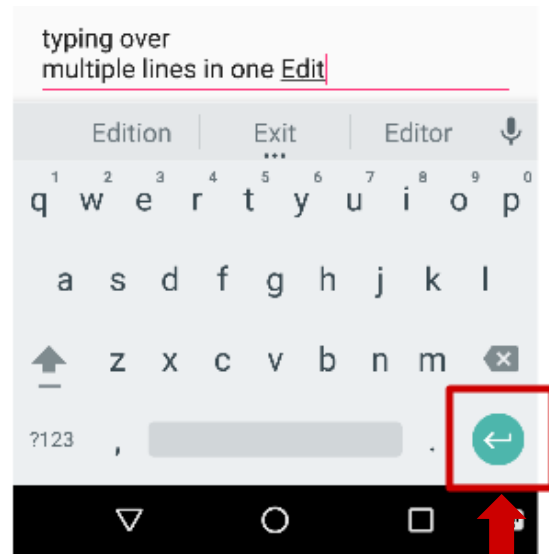
Getting text

- Get the EditText object for the EditText view

```
EditText simpleEditText =
    (EditText) findViewById(R.id.edit_simple);
```

- Retrieve the CharSequence and convert it to a string

```
String strValue =
    simpleEditText.getText().toString();
```

Commonput typesinput types

- textShortMessage—Limit input to 1 line

- textCapSentences—Set keyboard to caps at beginning of sentences

- textAutoCorrect—Enable autocorrecting

- textPassword—Conceal typed characters

- textEmailAddress—Show an @ sign on the keyboard

- phone—numeric keyboard for phone numbers

```
android:inputType="phone"

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

User Input Controls

Responding to button taps

- *In your code*: Use `OnClickListener` event listener.
- *In XML*: use `android:onClick` attribute in the XML layout:

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

android:onClick

Buttons

Button

- View that responds to clicking or pressing

- Usually text or visuals indicate what will happen when it is pressed

- Views: Button > ToggleButton, ImageView > FloatingActionButton (FAB)

- State: normal, focused, disabled, pressed, on/off

- Visuals: raised, flat, clipart, images, text

Setting listener with onClick callback

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

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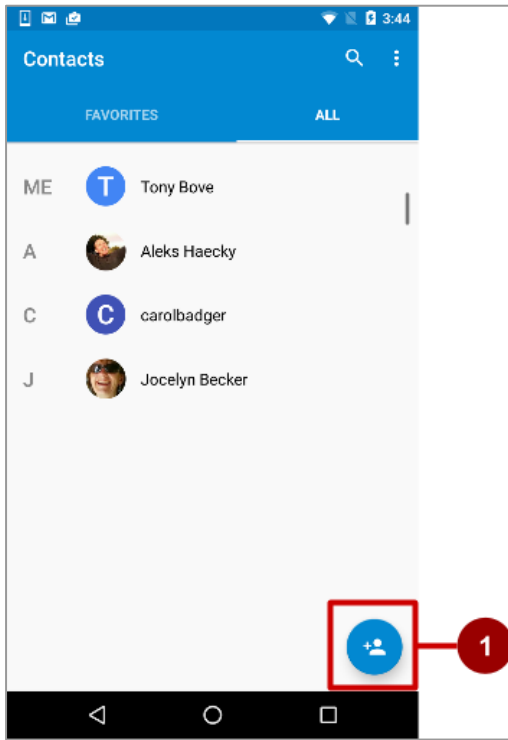
Floating Action Buttons (FAB)

- Raised, circular, floats above layout
- Primary or "promoted" action for a screen
- One per screen

For example:

Add Contact button in Contacts app

Using FABs

- Add design support library to build.gradle

```
compile 'com.android.support:design:a.b.c'
Latest:
implementation 'com.google.android.material:material:1.2.0-alpha04'
```

- Layout

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

Button image assets

1. Right-click app/res/drawable

2. Choose **New > Image Asset**

3. Choose **Action Bar and Tab Items** from drop down menu

4. Click the **Clipart:** image (the Android logo)

Experiment:
2. Choose **New > Vector Asset**

Demo of Floating Action Button

1. Add dependencies

2. Create a FAB in the activity_main.xml

3. Create images asset to add to FAB

4. In Main_Activity.java extract the FAB view and add listerner to perform the action.

User Input Controls

Animation

Animation is the process of creating motion and shape change.

Animation in android is possible from many ways. one easy and widely used way of making animation called tweened animation is discussed here.

Resources

1. In resources create an animation file res->anim->.xml

2. File is with .xml format

3. Need to add the animation codes for scale or rotate etc.

4. Scale, Rotate, Alpha,Translate

| **Android Developer Fundamentals** **User Input Controls** *This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License* 32

Scale

```
<scale xmlns:android="http://schemas.android.com/apk/res/android"
        android:fromXScale="0.5"
        android:toXScale="3.0"
       android:fromYScale="0.5"
       android:toYScale="3.0"
       android:duration="5000"
       android:pivotX="50%"
       android:pivotY="50%" >
</scale>
```

Rotate

```xml
<rotate xmlns:android="http://schemas.android.com/apk/res/and
        android:fromDegrees="0"
        android:toDegrees="360"
        android:pivotX="50%"
        android:pivotY="50%"
        android:duration="5000" >
</rotate>
```

Fading

Appear:
```
<alpha android:fromAlpha="0"
        android:toAlpha="1"
        android:duration="2000" >
</alpha>
```

Disappear:
```
<alpha android:startOffset="2000"
 android:fromAlpha="1"
 android:toAlpha="0"
android:duration="2000" >
</alpha>
```

Blink

```xml
<alpha android:fromAlpha="0.0"
       android:toAlpha="1.0"
android:interpolator="@android:anim/accelerate_interpolator"
       android:duration="600"
       android:repeatMode="reverse"
       android:repeatCount="infinite"/>
```

User Input Controls

Move

```
<translate android:fromXDelta="0%p"
          android:toXDelta="75%p"
          android:duration="800" />
```

Slide

```
<scale android:duration="500"
 android:fromXScale="1.0"
android:fromYScale="1.0"
android:interpolator="@android:anim/linear_interpolator"
android:toXScale="1.0"
android:toYScale="0.0" />
```

User Input Controls

Animation Object creation

```java
Animation anim= AnimationUtils.loadAnimation(

                getApplicationContext(),R.animation_file);


View.startAnimation(anim); //start animation with
                           animation object
```

Demo
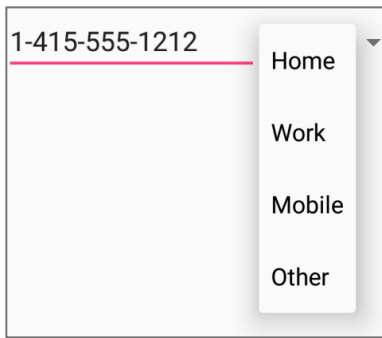
1. Create an image, text and button.

2. Apply all animation features on the image,text and button.

Making Choices

So many choices!

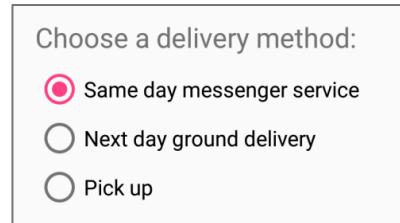- Checkboxes

- Radio buttons

- Toggles

- Spinner

User Input Controls

Checkboxes, radio buttons and toggles

Checkboxes
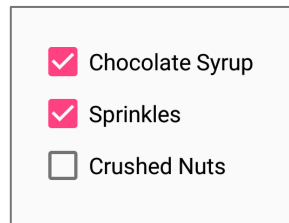
- User can select any number of choices
- Checking one box does not uncheck another
- Users expect checkboxes in a vertical list
- Commonly used with a submit button
- Every checkbox is a view and can have an onClick handler

☑ Chocolate Syrup

☑ Sprinkles

☐ Crushed Nuts

Radio buttons

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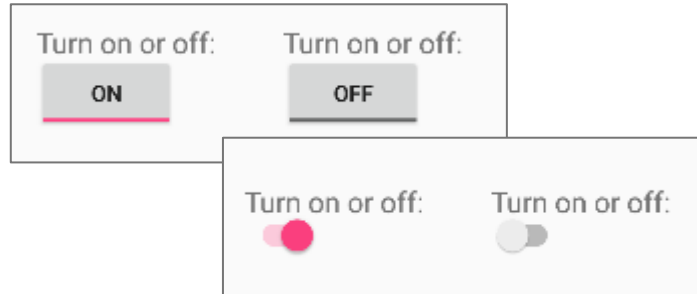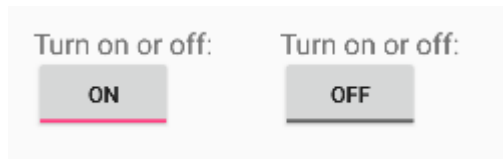
Choose a delivery method:

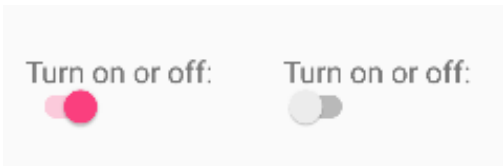- ⦿ Same day messenger service
- ◯ Next day ground delivery
- ◯ Pick up

45

Toggle buttons and switches

- User can switch between 2 exclusive states (on/off)
- Use `android:onClick`+callback—or handle clicks in code

Toggle buttons

Switches

Chip and ChipGroup

Chip

Chips are basically a text displayed in a rounded background. These are checkable and can contain icons as well. **Chips** are a newer and stylised form of RadioButton.

```
<com.google.android.material.chip.Chip
style="@style/Widget.MaterialComponents.Chip.Choice"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
app:chipText="Choice" />
```

Android ChipGroup

Similar to RadioGroups, ChipGroups are used to hold Chips

```
<com.google.android.material.chip.ChipGroup
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_marginTop="16dp">
```

ChipGroups

ChipGroups by default spaces the Chips present inside it.

Few of the XML attributes that can be used with ChipGroups are:

app:chipSpacing: To set a custom spacing value between the chips, both horizontally and vertically.

app:chipSpacingHorizontal
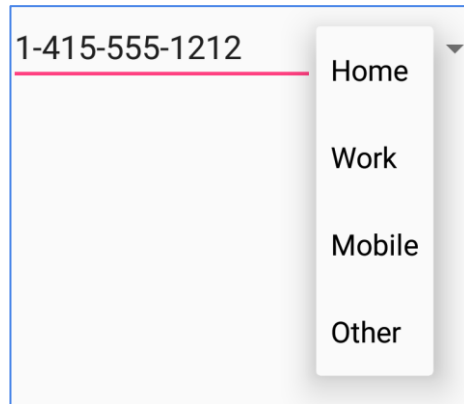
app:chipSpacingVertical

app:singleSelection – Setting this as true allows only one of the chips to be checked.

app:singleLine – Sets all the chips present, in a single line only.

Spinners

Spinners

- Quick way to select value from a set

- Drop-down list shows all values, user can select only one

- Spinners scroll automatically if necessary

- Use the `Spinner` class.

52

Implementing a spinner

1. Create Spinner UI element in the XML layout

2. Define spinner choices in an array

3. Create Spinner and set onItemSelectedListener

4. Create an adapter with default spinner layouts

5. Attach the adapter to the spinner

6. Implement `onItemSelectedListener` method

Create spinner XML

In layout XML file

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

Define array of spinner choices

In arrays.xml resource file

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

Create spinner and attach listener

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

// In onCreate()

Spinner spinner = (Spinner) findViewById(R.id.label_spinner);
if (spinner != null) {
        spinner.setOnItemSelectedListener(this);
}
```
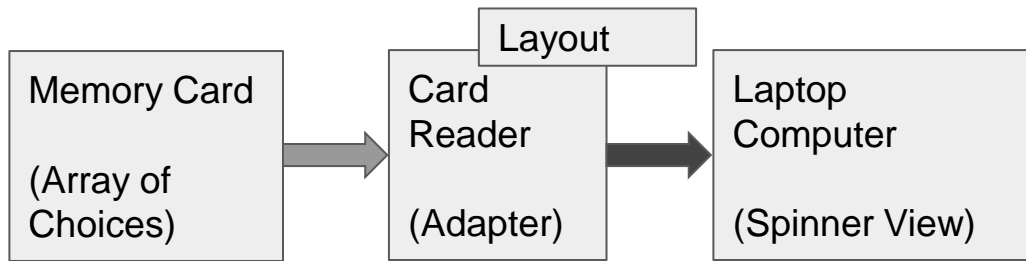
What is an adapter?

An adapter is like a bridge, or intermediary, between two incompatible interfaces

For example, a memory card reader acts as an adapter between the memory card and a laptop

57

Create adapter

Create ArrayAdapter using string array
and default spinner layout

```
ArrayAdapter<CharSequence> adapter =
  ArrayAdapter.createFromResource(
        this, R.array.labels_array,
        // Layout for each item
        android.R.layout.simple_spinner_item);
```

Attach the adapter to the spinner

- Specify the layout for the drop down menu

```
adapter.setDropDownViewResource(
    android.R.layout.simple_spinner_dropdown_item);
```

- Attach the adapter to the spinner

```
spinner.setAdapter(adapter);
```

User Input Controls

59

Implement onItemSelectedListener

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

    public void onItemSelected(AdapterView<?> adapterView,
        View view, int pos, long id) {
            String spinner_item =
                adapterView.getItemAtPosition(pos).toString();
            // Do something here with the item
    }

    public void onNothingSelected(AdapterView<?> adapterView) {
        // Do something
    }
```
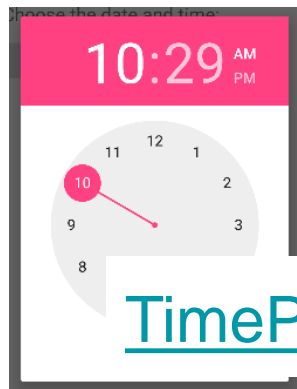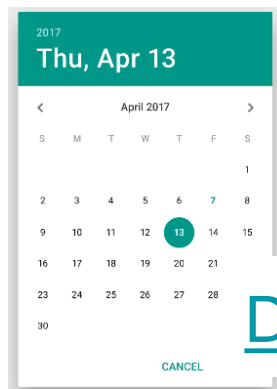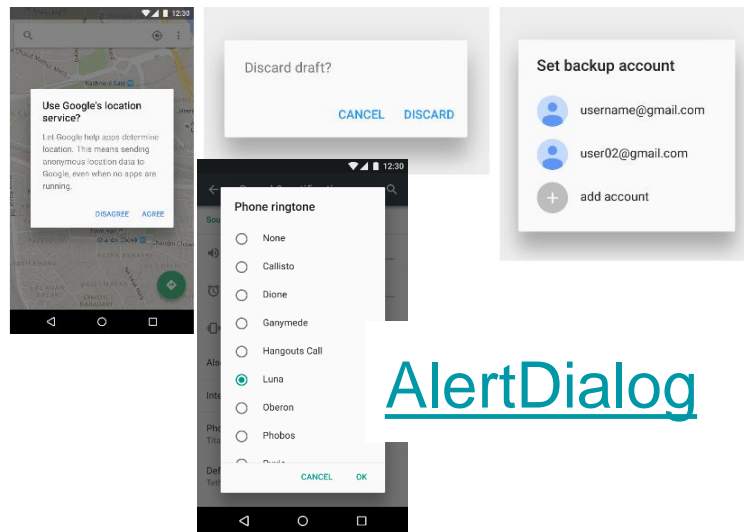
Dialogs

Dialogs

- **Dialog** appears on top, interrupting the flow of activity

- Require user action to dismiss

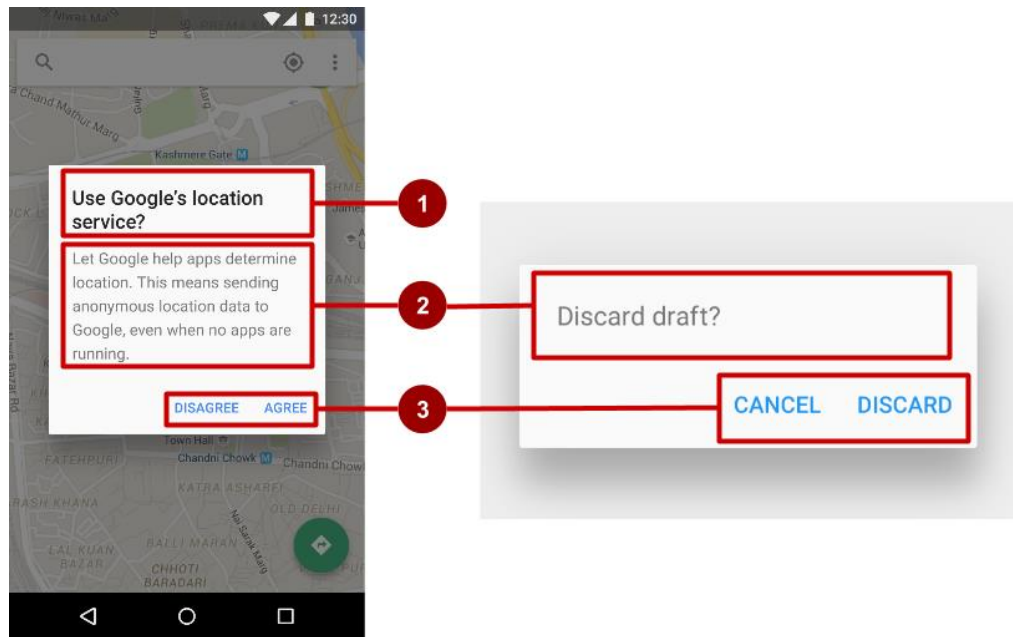AlertDialog

TimePickerDialog

DatePickerDialog

AlertDialog

AlertDialog can show:

1. Title (optional)

2. Content area

3. Action buttons

Build the AlertDialog

Use `AlertDialog.Builder` to build a standard alert dialog and set attributes:

```
public void onClickShowAlert(View view) {

AlertDialog.Builder alertDialog = new
                      AlertDialog.Builder(MainActivity.this);

alertDialog.setTitle("Connect to Provider");

alertDialog.setMessage(R.string.alert_message);

...
```

User Input Controls

Add the button actions

- `alertDialog.setPositiveButton()`

- `alertDialog.setNeutralButton()`

- `alertDialog.setNegativeButton()`

alertDialog code example

```
alertDialog.setPositiveButton(

    "OK", newDialogInterface.OnClickListener() {

        public void onClick(DialogInterface dialog, int which) {

        // User clicked OK button.

    }

});
```
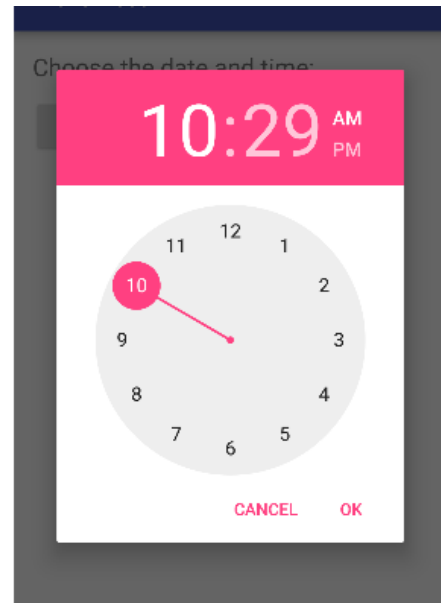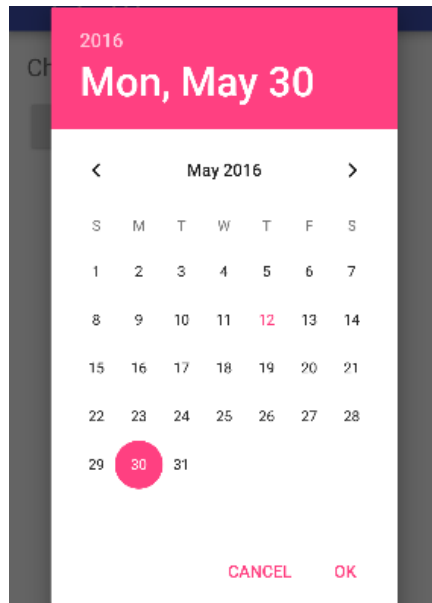
Same pattern for setNegativeButton() and setNeutralButton()
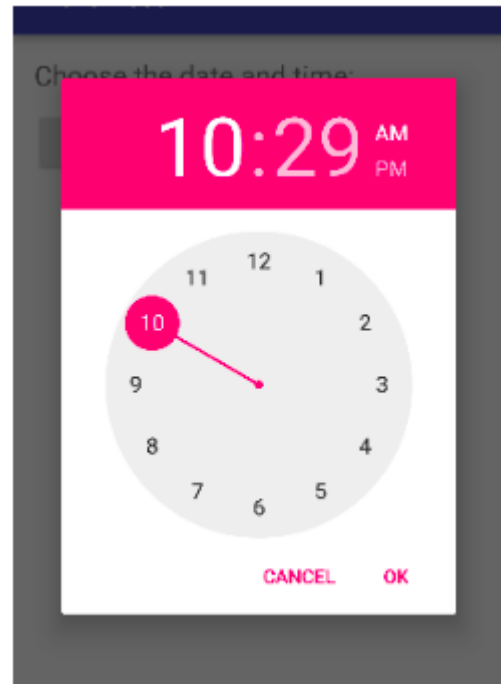
Pickers

Pickers

- ## DatePickerDialog

- ## TimePickerDialog

Pickers use fragments

- Use DialogFragment to show a picker

- DialogFragment is a window that floats on top of activity's window

User Input Controls

Introduction to fragments

- A fragment is like a mini-activity within an activity

  - Manages its own own lifecycle.

  - Receives its own input events.

- Can be added or removed while parent activity is running

- Multiple fragments can be combined in a single activity

- Can be reused in multiple activities

Creating a date picker dialog

1. Add a blank fragment that extends `DialogFragment` and implements `DatePickerDialog.OnDateSetListener`

2. In `onCreateDialog()` initialize the date and return the dialog

3. In `onDateSet()` handle the date

4. In Activity show the picker and add a method to use the date

Creating a time picker dialog

1. Add a blank fragment that extends `DialogFragment` and implements `TimePickerDialog.OnTimeSetListener`

2. In `onCreateDialog()` initialize the time and return the dialog

3. In `onTimeSet()` handle the time

4. In Activity, show the picker and add a method to use the time

Common Gestures

Touch Gestures

Touch gestures include:

- long touch
- double-tap
- fling
- drag
- scroll
- pinch

Don't depend on touch gestures for app's basic behavior!

Android Developer Fundamentals

User Input Controls

Detect gestures

Classes and methods are available to help you handle gestures.

- GestureDetectorCompat class for common gestures

- MotionEvent class for motion events

Detecting all types of gestures

1. Gather data about touch events.

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

Read more about how to handle gestures in the
Android developer documentation