Activities and Intents

Contents

- Activities
- Defining an `Activity`
- Starting a new `Activity` with an `Intent`
- Passing data between activities with extras
- Navigating between activities

These slides are partially based on the material that Google provides for the course ***Android Developer Fundamentals***

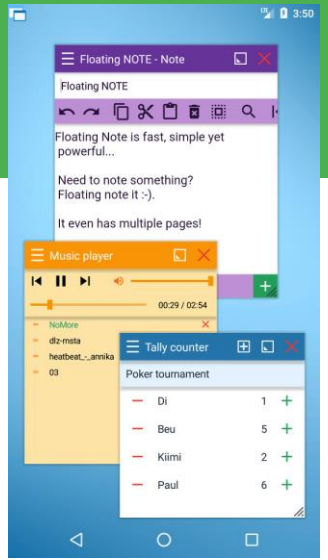
https://developer.android.com/courses/fundamentals-training/overview-v2

Activities (high-level view)

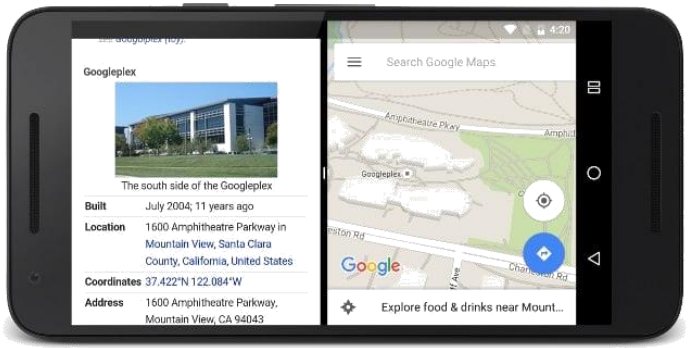
What is an Activity?

- An `Activity` is an **application component**
- **Represents one window**
  - one hierarchy of views
- **Typically fills the screen**
  - but can be embedded in other Activity
  - or a appear as floating window
- A Java class (typically one Activity in one file)

What is an Activity?



- An `Activity` is an **application component**
- **Represents one window**
  - one hierarchy of views
- **Typically fills the screen**
  - but can be embedded in other Activity
  - or a appear as floating window
- A Java class (typically one Activity in one file)

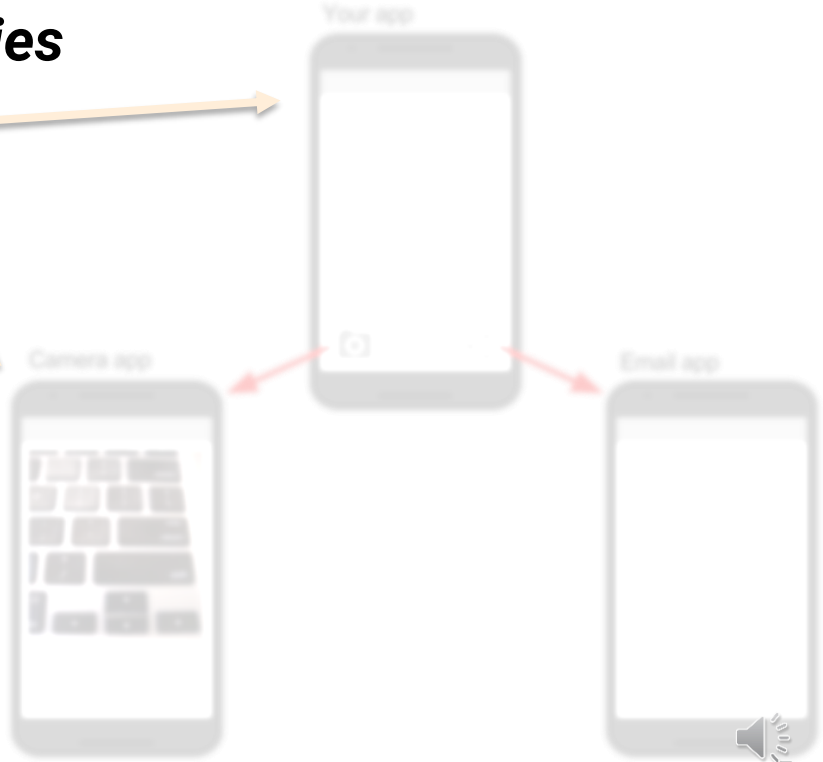


What is an Activity?

- An ***activity*** **represents a single screen** in the app with an interface the user can interact with
  - e.g. **an email app** might have **3 activities**
    - one to shows a list of received emails
    - one to compose an email
    - one to read individual messages

What does an Activity do?

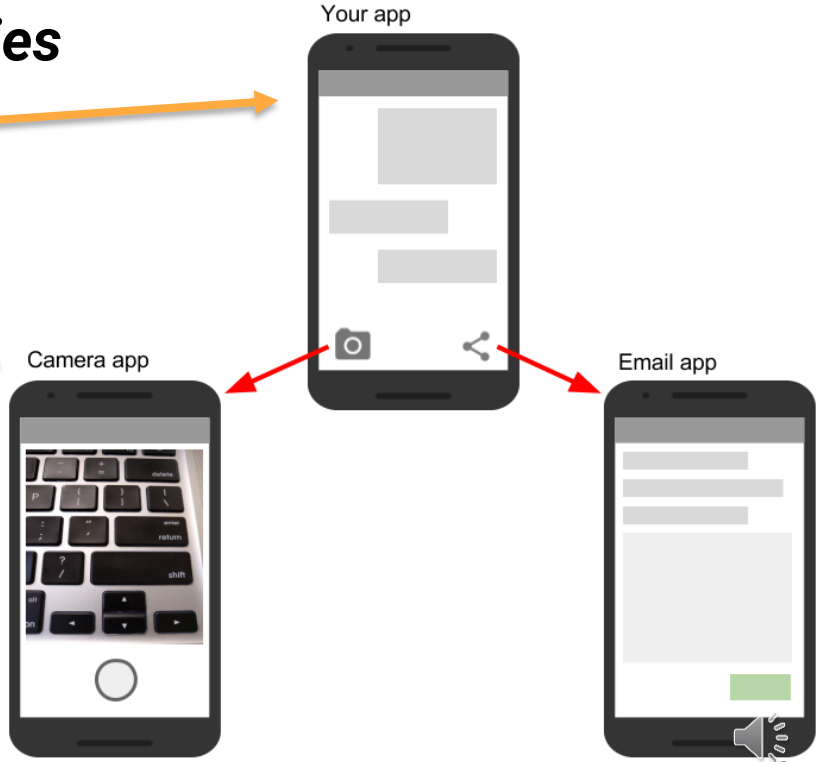
- **Apps** are often a ***collection of activities***
  - that you create yourself
  - that you reuse from other apps



- Handles user interactions, such as
  - button clicks
  - text entry
  - login verification

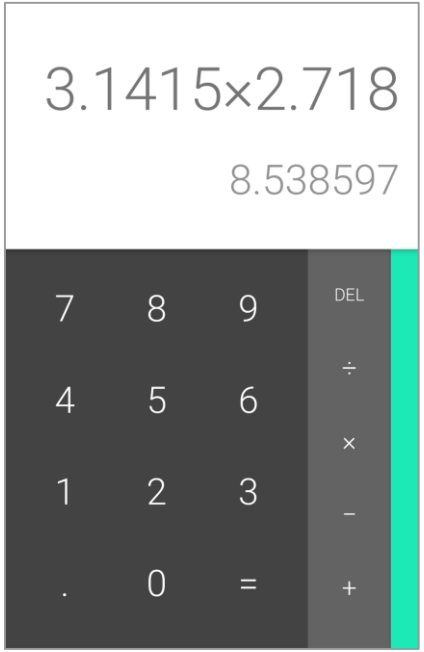
What does an Activity do?

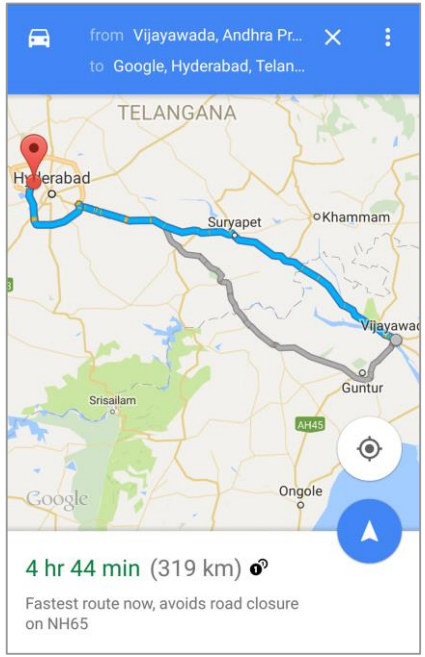
- **Apps** are often a ***collection of activities***
  - that you create yourself
  - that you reuse from other apps



- Handles user interactions, such as
  - button clicks
  - text entry
  - login verification

Examples of activities



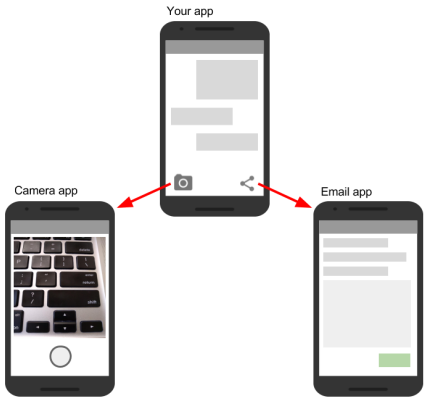


Apps and activities

- ***First Activity user sees*** is typically called **"main activity"**

- Activities **are *loosely tied*** together to make up an app

- Activities can be organized in **parent-child relationships** in the Android manifest **to aid navigation**

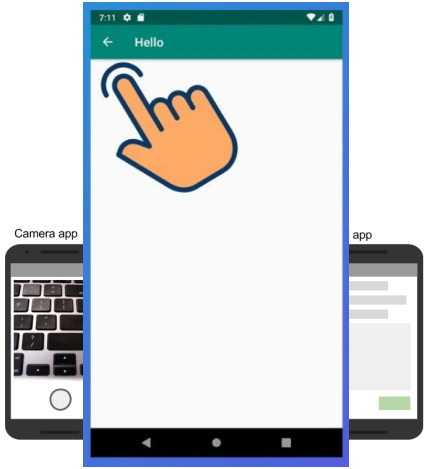


Apps and activities

- ***First Activity user sees*** is typically called **"main activity"**

- Activities **are *loosely tied*** together to make up an app

- Activities can be organized in **parent-child relationships** in the Android manifest **to aid navigation**

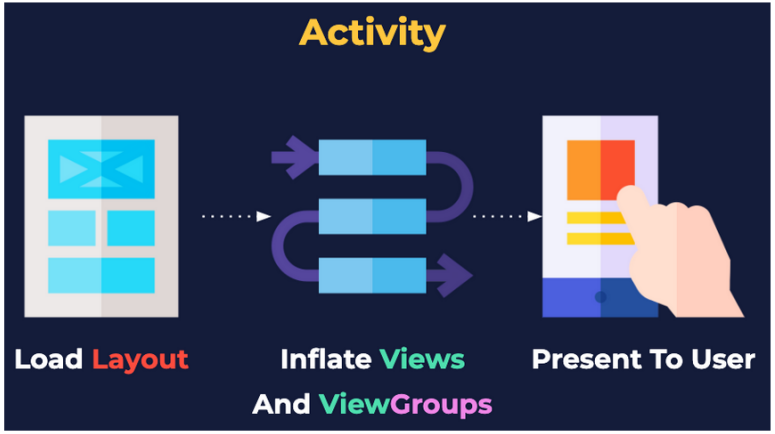


What does an Activity do?

- Has a life cycle: an activity is
  - Created
  - Started
  - Runs
  - Paused
  - Resumed
  - Stopped
  - Destroyed

Layouts and Activities

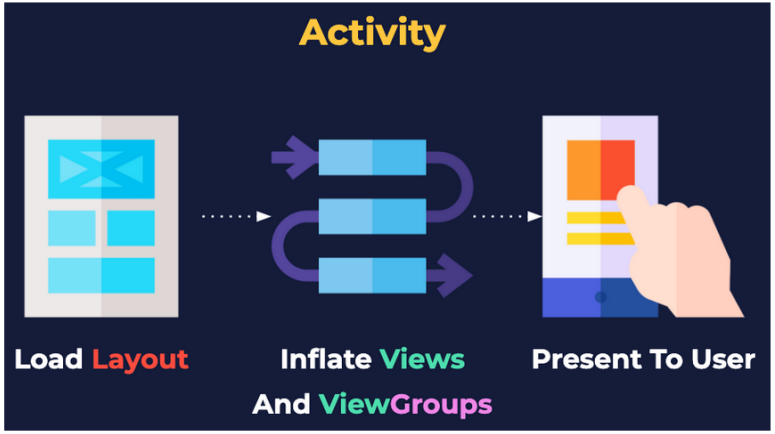
- An `Activity` typically has a UI layout
- Layout is usually defined in one or more XML files
- `Activity` "inflates" layout as part of being created




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

Layouts and Activities

- An `Activity` typically has a UI layout
- Layout is usually defined in one or more XML files
- `Activity` "inflates" layout as part of being created




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

Implementing Activities

Implement new activities

When ***creating a new project***

or add a new Activity to an app by choosing File > New > Activity

the **wizard automatically *performs* the following steps**:

1. Define layout in XML
2. Define `Activity` Java class
   - extends `AppCompatActivity`
3. Connect `Activity` with Layout
   - Set content view in `onCreate()`
4. Declare `Activity` in the Android manifest

1. Define layout in XML

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <TextView
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="Let's Shop for Food!" />
</RelativeLayout>
```

2. Define Activity Java class

When ***creating a new project*** the **MainActivity** is, by default, a ***subclass of the AppCompatActivity class***

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
}
```

This **allows to use up-to-date Android app** features such as the app bar and Material Design while still **enabling the app to be compatible** with devices running **older versions of Android.**

2. Define Activity Java class

The **first task** in the implementation of an Activity subclass **is to implement the standard Activity lifecycle callback methods** (such as onCreate()) to handle the state changes for your Activity.

```
public class MainActivity extends AppCompatActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);
   }
}
```

These state changes include things such as when the Activity is **created**, **stopped**, **resumed**, or **destroyed**.

2. Define Activity Java class

The **one required callback** that an app must implement is the **onCreate()** method.

The ***system*** ***calls this method when it creates the Activity***, and all the essential components of your Activity should be initialized here.

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

3. Connect activity with layout

The onCreate() method calls **setContentView()** with the path to a layout file.

The **system** **creates all the initial views** from the specified layout and adds them to your Activity. This is often referred to as ***inflating the layout***.

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Resource is layout in this XML file

4. Declare activity in Android manifest

Each Activity in an app must be declared in the AndroidManifest.xml file with the <activity> element, inside the <application> section.

The only required attribute is `android:name`, which specifies the class name for the Activity (such as MainActivity)

```
<activity android:name=".MainActivity" >
   <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
   </intent-filter>
</activity>
```

4. Declare activity in Android manifest

Each Activity in an app must be declared in the AndroidManifest.xml file with the <activity> element, inside the <application> section.

The only required attribute is `android:name`, which specifies the class name for the Activity (such as MainActivity)

```
<activity android:name=".MainActivity" >
   <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
   </intent-filter>
</activity>
```

4. Declare main activity in manifest

`MainActivity` **needs to include `intent-filter`** to start from launcher
The `<action>` element specifies that this is the "main" entry point to the app

```
<activity android:name=".MainActivity" >

    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>

</activity>
```

only the MainActivity should include the "main" action

4. Declare main activity in manifest

`MainActivity` **needs to include `intent-filter`** to start from launcher
The `<action>` element specifies that this is the "main" entry point to the app

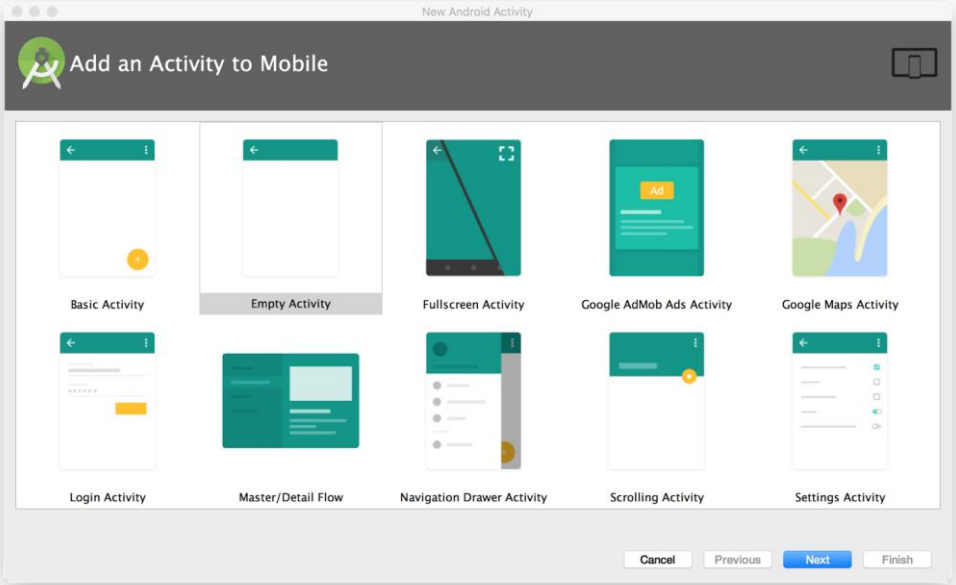
```
<activity android:name=".MainActivity" >

    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>

</activity>
```

only the MainActivity should include the "main" action

Add Activities to App

Each Activity of an app and its associated layout file is supplied by an Activity template in Android Studio such as Empty Activity or Basic Activity.

You can add a new Activity to your project by choosing
File > New > Activity

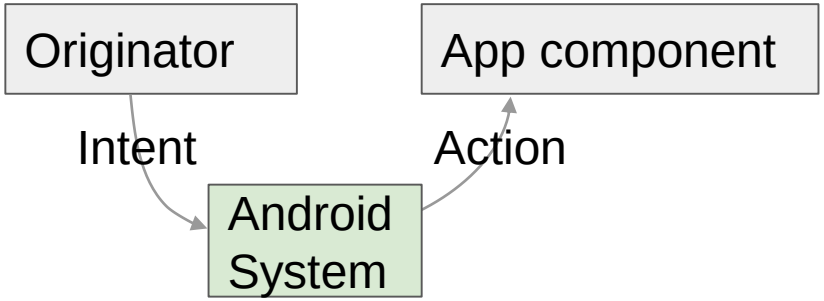


Intents

What is an intent?

An `Intent` is ***a description of an operation to be performed***.

An `Intent` is an object used to request an action from another app component via the Android system.



Starting the main activity

1. When the app is **first started** from the **device home screen**
2. the **Android runtime sends an Intent to the app** to start the app's main activity

   the one defined with the MAIN action and the LAUNCHER category in the AndroidManifest.xml file

```
<activity android:name=".MainActivity" >

   <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
   </intent-filter>

</activity>
```

What can intents do?

- Start an `Activity`
  - A button click starts a new `Activity` for text entry
    - pass data between one activity and another
  - Clicking Share opens an app that allows you to post a photo
- Start a `Service`
  - Initiate downloading a file in the background
- Deliver `Broadcast`
  - The system informs everybody that the phone is now charging

Explicit and implicit intents

Explicit Intent

- Starts a specific `Activity`
  - e.g. Request tea with milk delivered by a specific Cafe
  - `Main activity` starts the `ViewShoppingCart Activity`

Implicit Intent

- Asks system to find an `Activity` that can handle this request
  - e.g. Find an open store that sells green tea
  - Clicking Share opens a chooser with a list of apps

Explicit and implicit intents

Explicit Intent

- Starts a specific `Activity`
  - e.g. Request tea with milk delivered by a specific Cafe
  - `Main activity` starts the `ViewShoppingCart Activity`

Implicit Intent

- Asks system to find an `Activity` that can handle this request
  - e.g. Find an open store that sells green tea
  - Clicking Share opens a chooser with a list of apps

Starting Activities

Start an Activity with an explicit intent

To start a **specific `Activity`**, use **an explicit `Intent`**

1. Create an `Intent`
   - `Intent intent = new Intent(this, ActivityName.class);`
2. Use the `Intent` to start the `Activity`
   - `startActivity(intent);`

```
Example:
Intent messageIntent = new Intent(this, ShowMessageActivity.class);
startActivity(messageIntent);
```

Start an Activity with an explicit intent

To start a **specific `Activity`**, use **an explicit `Intent`**

1. Create an `Intent`
   - `Intent intent = new Intent(this, ActivityName.class);`
2. Use the `Intent` to start the `Activity`
   - `startActivity(intent);`

```
Example:
Intent messageIntent = new Intent(this, ShowMessageActivity.class);
startActivity(messageIntent);
```

Start an Activity with implicit intent

To **ask Android to find an `Activity`** to handle your request, use **an implicit `Intent`**

1. Create an `Intent`
   - `Intent intent = new Intent(action, uri);`
2. Use the `Intent` to start the `Activity`
   - `startActivity(intent);`

Implicit Intents - Examples

Show a web page

```
Uri uri = Uri.parse("http://www.google.com");
Intent it = new Intent(Intent.ACTION_VIEW, uri);
startActivity(it);
```

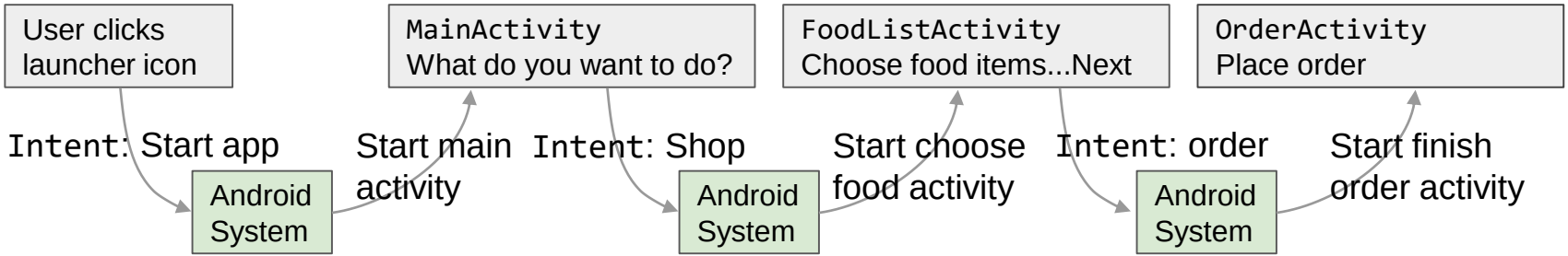
Dial a phone number

```
Uri uri = Uri.parse("tel:8005551234");
Intent it = new Intent(Intent.ACTION_DIAL, uri);
startActivity(it);
```

https://developer.android.com/reference/android/content/Intent.html

How Activities Run

- All `Activity` instances are managed by the Android runtime
- Started by an "`Intent`", a message to the Android runtime to run an activity



Sending and Receiving Data

Sending data with intents

In addition to **open a new activity, with an intent** it is possible **to pass data** from one Activity to another.

In particular, it is possible to use **Intent data** or **Intent extras**

- **Data**: **one piece** of information whose data location can be **represented by an URI**
- **Extras**: **one or more pieces** of information as a **collection of key-value pairs**

There are several key differences between data and extras that determine which you should use

Intent data

The **Intent data** can hold **only one piece** of information: a **URI** representing the location of the data you want to operate on.

That URI could be:

- a web page URL (http://),
- a telephone number (tel://),
- a geographic location (geo://) or
- any other custom URI you define.

A Uniform Resource Identifier (URI) is a compact string of characters for identifying an abstract or physical resource

https://www.ietf.org/rfc/rfc2396.txt

When use Intent data

Use the Intent data field when:

- you only have **one piece of information** that you need to send to the started Activity
- that **information** is a data location that can be **represented by a URI**

Extras Data

Intent extras are for **any other arbitrary data** you want to pass to the started Activity.

Intent extras **are stored in a Bundle** object as **key and value pairs**.

A **Bundle is a map**, optimized for Android, in which

- a **key** is a **string**
- a **value** can be **any primitive** or **object type**
  - objects must implement the Parcelable interface

Extras Data

To put data into the Intent extras you can

- **use** any of the Intent class **putExtra()** methods

- **create** your **own Bundle**

  + put **the Bundle** into the Intent with **putExtras()**.

When use Extras Data

Use the **Intent extras**:

- If you want **to pass more than one piece of information** to the started Activity.
- If any of the information you want to pass **is not expressible by a URI**.

NOTE! **Intent data** and **extras** **are not exclusive**;

- you **can use data for a URI** ***and***
- **extras for any additional information** the started Activity needs to process the data in that URI.

Passing data from one Activity to another

For **sending data** to an `Activity`:

1. **Create** the `Intent` object
2. **Put data** or **extras** into that `Intent`
3. **Start the new `Activity`**

For **receiving data** from an `Activity`:

1. **Get** the `Intent` object, the `Activity` was started with
2. **Retrieve the data** or extras from the `Intent` object

Sending Data - Create the Intent object

Step 1: **Create** the **`Intent` object**

```
Intent messageIntent = new Intent(this,
ShowMessageActivity.class);
```

Passing data from one Activity to another

For **sending data** to an `Activity`:

1. **Create** the `Intent` object
2. **Put** **data** or **extras** into that `Intent`
3. **Start the new `Activity`**

For **receiving data** from an `Activity`:

1. **Get** the `Intent` object, the `Activity` was started with
2. **Retrieve the data** or extras from the `Intent` object

Sending Data - Put data into that Intent

Step 2: **Put data** into that **Intent**

Use the setData() method with a URI object to add it to the Intent.

Some examples of using setData() with URIs:

```
// A web page URL
intent.setData(Uri.parse("http://www.google.com"));

// a Sample file URI
intent.setData(Uri.fromFile(new File("/sdcard/sample.jpg")));
```

Sending Data - Start the new Activity

Keep in mind that

- the data field **can only contain a single URI**
- If you **call setData() multiple times** **only the last value is used**
- You should use **Intent extras to include additional information (including additional URIs**)

Step 3: After you've added the data, you can start the new Activity with the Intent:

```
startActivity(messageIntent);
```

Passing data from one Activity to another

For **sending data** to an `Activity`:

1. **Create** the `Intent` object
2. **Put data** or **extras** into that `Intent`
3. **Start the new `Activity`**

For **receiving data** from an `Activity`:

1. **Get** the `Intent` object, the `Activity` was started with
2. **Retrieve the data** or extras from the `Intent` object

Sending Extras (with multiple putExtra)

Step 1: **Create the `Intent`** object (***as seen for data***)

Step 2: **Put extras** into that `Intent`

Use a putExtra() method with a **key** to put data into the Intent extras. The Intent class defines **many putExtra() methods for different kinds** of data, for example:

- `putExtra(String name, int value)`
  ⇒ `intent.putExtra("level", 406);` manage an integer
- `putExtra(String name, String[] value)`
  ⇒ `String[] foodList = {"Rice", "Beans", "Fruit"};`
  `intent.putExtra("food", foodList);`

  I create the object which is the string object list that can we can use in the putExtra method

Step 3: `startActivity(messageIntent);`

How can I pass an **object of a custom type** from one Activity to another using the putExtra()?
https://stackoverflow.com/questions/2139134/how-to-send-an-object-from-one-android-activity-to-another-using-intents

Example

```
public static final String EXTRA_MESSAGE_KEY =
    "com.example.android.twoactivities.extra.MESSAGE";
```

Conventionally you define Intent extra keys as static variables with names that begin with EXTRA_. To guarantee that the key is unique, the string value for the key itself should be prefixed with your app's fully qualified class name.

```
Intent intent = new Intent(this, SecondActivity.class);
String message = "Hello Activity!";
intent.putExtra(EXTRA_MESSAGE_KEY, message);
startActivity(intent);
```

Sending Extras (with a Bundle)

Step 2 (alternative): **if lots of data**

- first create a bundle and
- pass the bundle

Bundle defines many "put" methods for different kinds of primitive data as well as objects that implement Android's Parcelable interface or Java's Serializable

Sending Extras (with a Bundle)

```
Bundle extras = new Bundle();
extras.putString(EXTRA_MESSAGE, "this is my message");
extras.putInt(EXTRA_POSITION_X, 100);
extras.putInt(EXTRA_POSITION_Y, 500);
```

After you've populated the Bundle, add it to the Intent with the putExtras() method (note the "s" in Extras):

```
messageIntent.putExtras(extras);

startActivity(intent);
```

Passing data from one Activity to another

For **sending data** to an `Activity`:

1. **Create** the `Intent` object
2. **Put data** or **extras** into that `Intent`
3. **Start the new `Activity`**

For **receiving data** from an `Activity`:

so we get also the data passed with the intent of the activity that we have created/we want to use

1. **Get** the `Intent` object, the `Activity` was started with
2. **Retrieve the data** or extras from the `Intent` object

Get data from intents

When you **start an Activity** with **an Intent**, **the started Activity has access to the Intent** and the data it contains.

To retrieve the Intent the Activity (or other component) was started with, use the getIntent() method:

```
Intent intent = getIntent();
```

Get data and extras from intents

Data: Use getData() to get the URI from that Intent:

- `getData();`
  ⇒ `Uri locationUri = intent.getData();` we have already setted in the intent with " intent.setData(...)"

Extras: Use one of the getExtra() methods to extract extra data out of the Intent object:

key value pair

- `int getIntExtra (String name, int defaultValue)`
  ⇒ `int level = intent.getIntExtra("level", 0);`

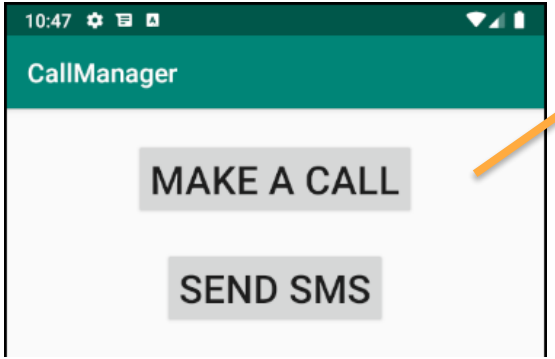
- `Bundle bundle = intent.getExtras();`
  ⇒ Get all the data at once as a bundle. we can retireve all the data like this

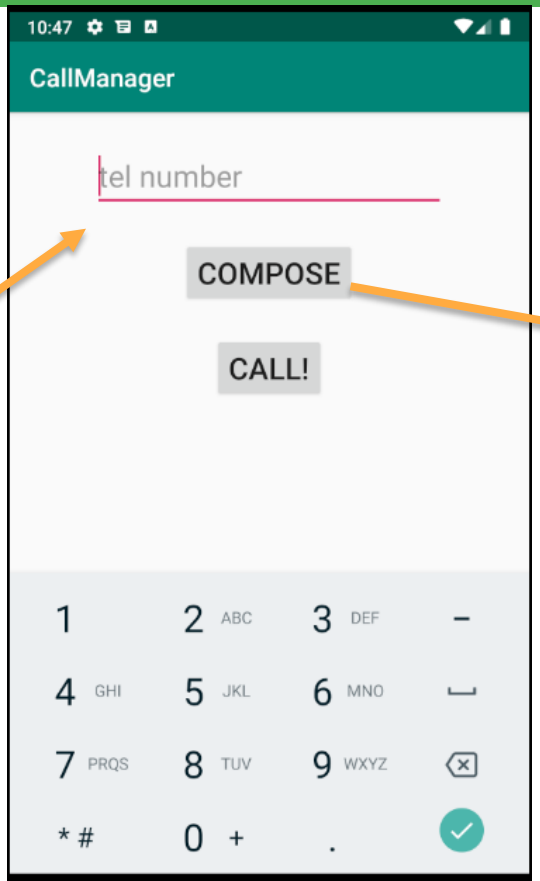
Calls and SMS Manager Exercise

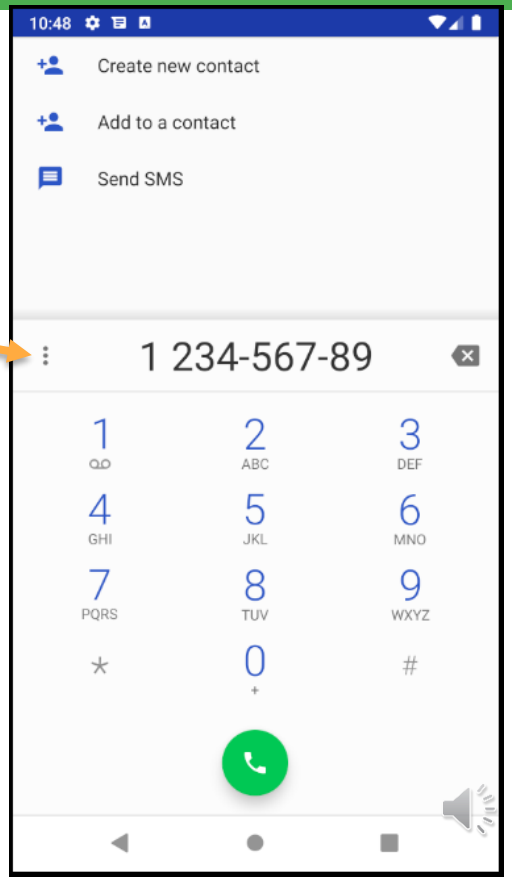
The goal of this exercise is to ***create an application*** that allows to:

1) **Make a direct call** to a specific telephone number
2) **Compose** a telephone number
3) **Send an SMS** to a telephone number
4) **Make a direct call** to a specific international telephone number
5) **Compose** an international telephone number

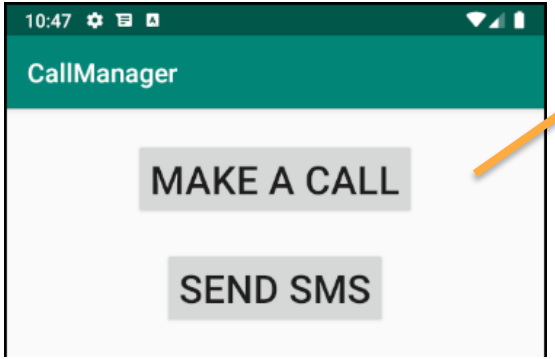
Calls and SMS Manager Exercise

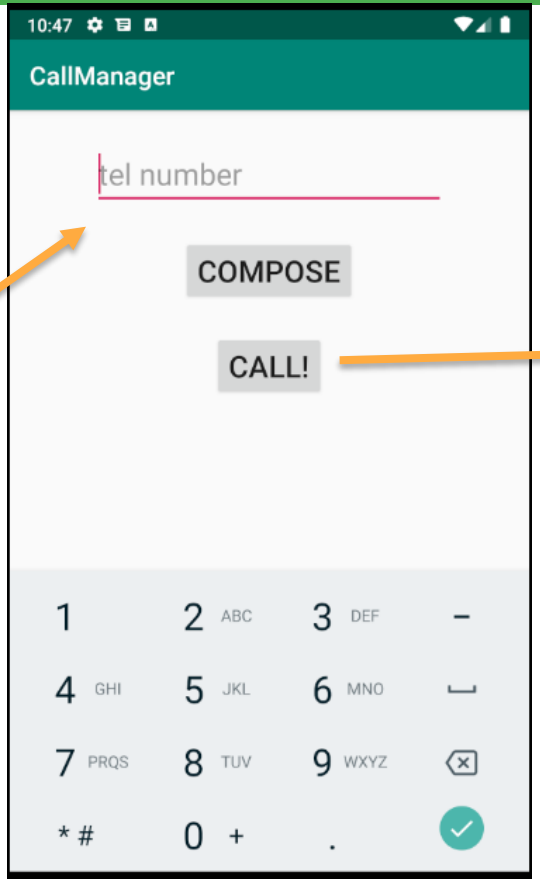


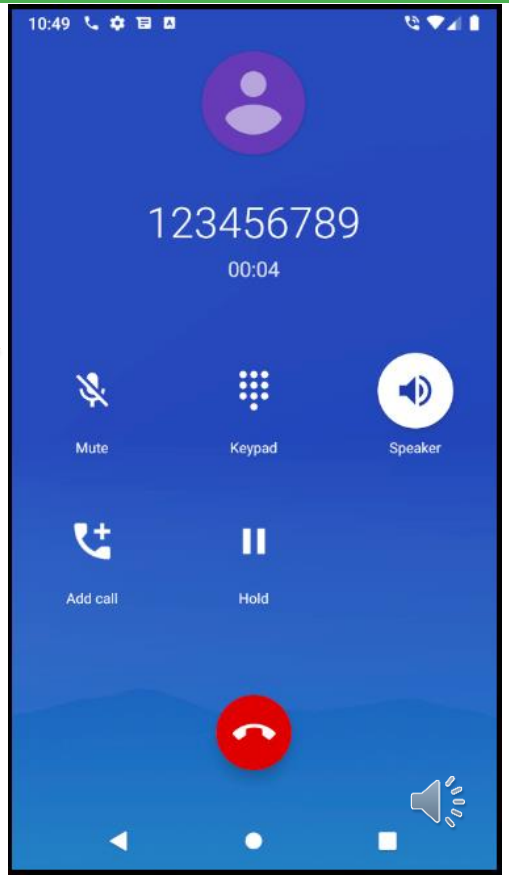




Calls and SMS Manager Exercise

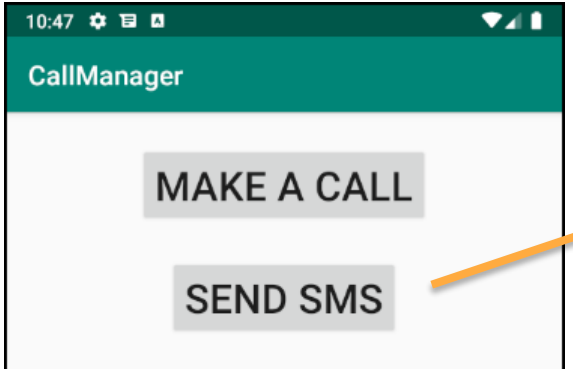


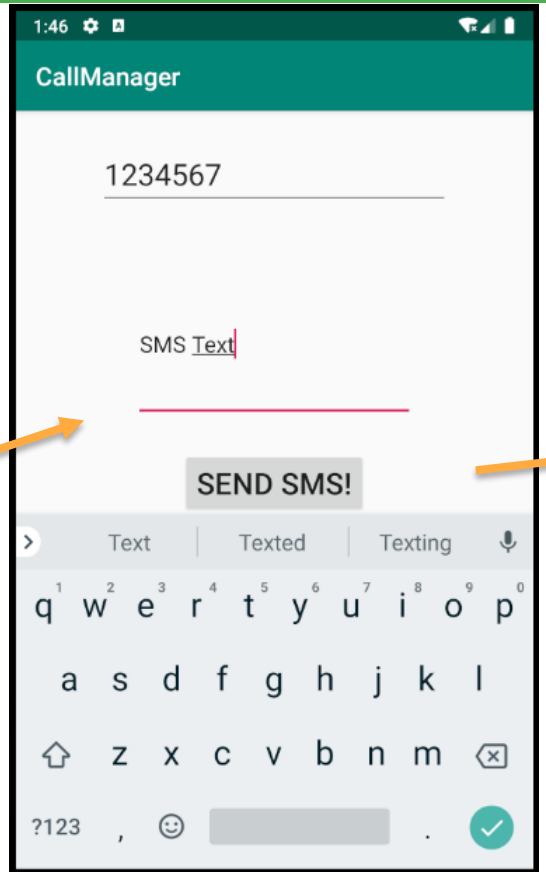


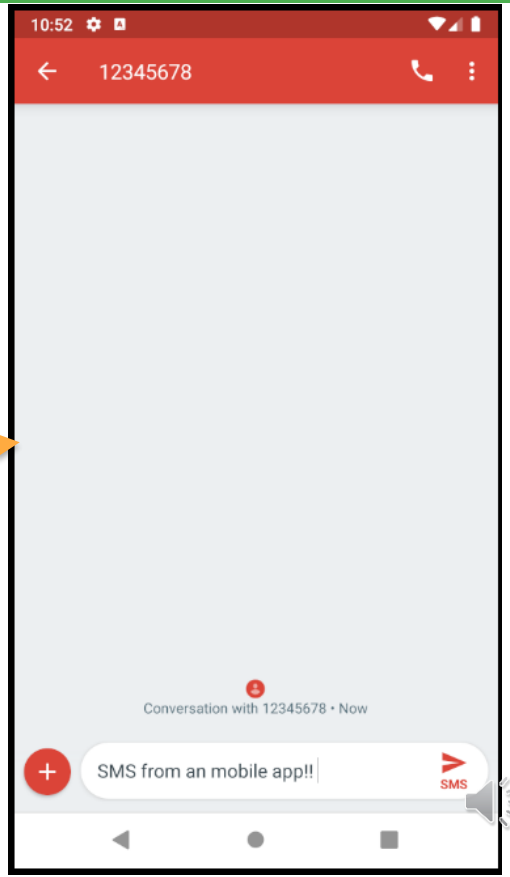


Warning: if you are using a real device you are making real calls!

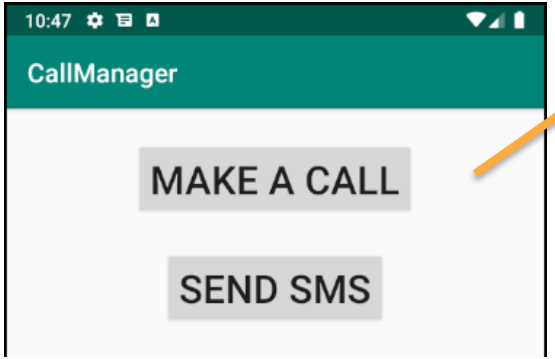
Calls and SMS Manager Exercise

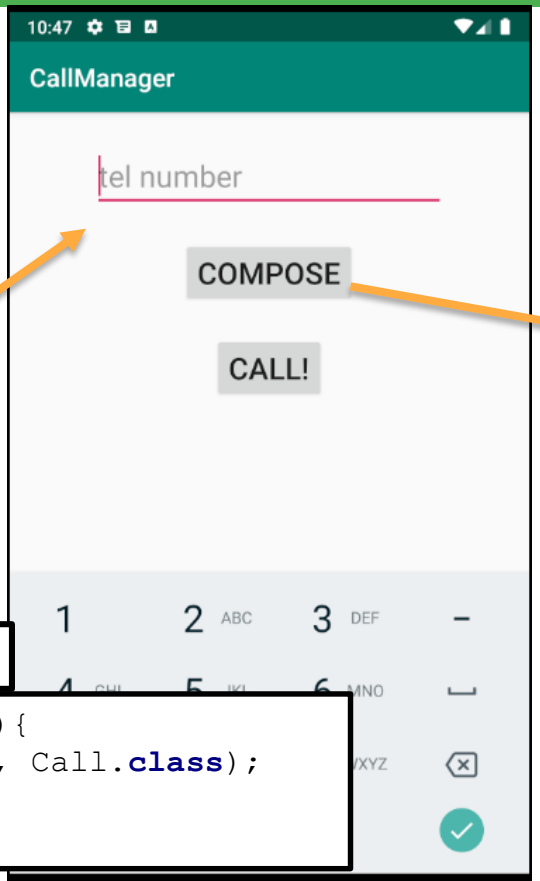


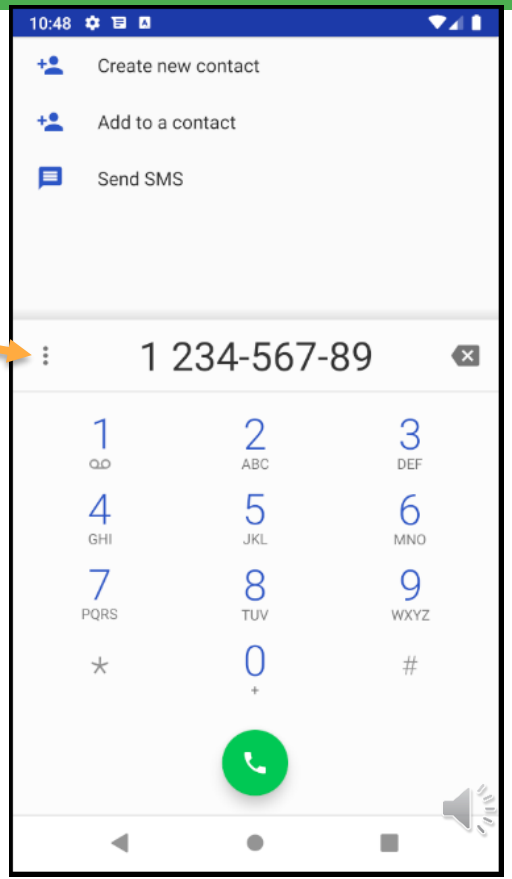




Calls and SMS Manager Exercise



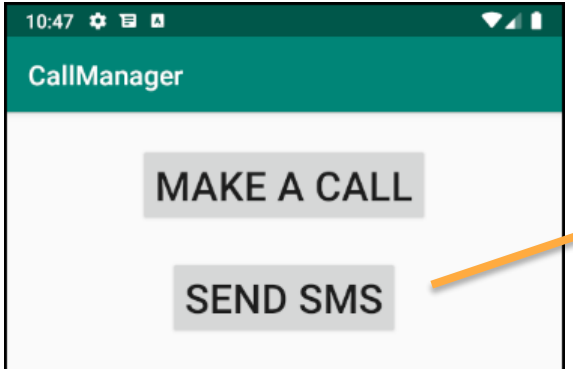


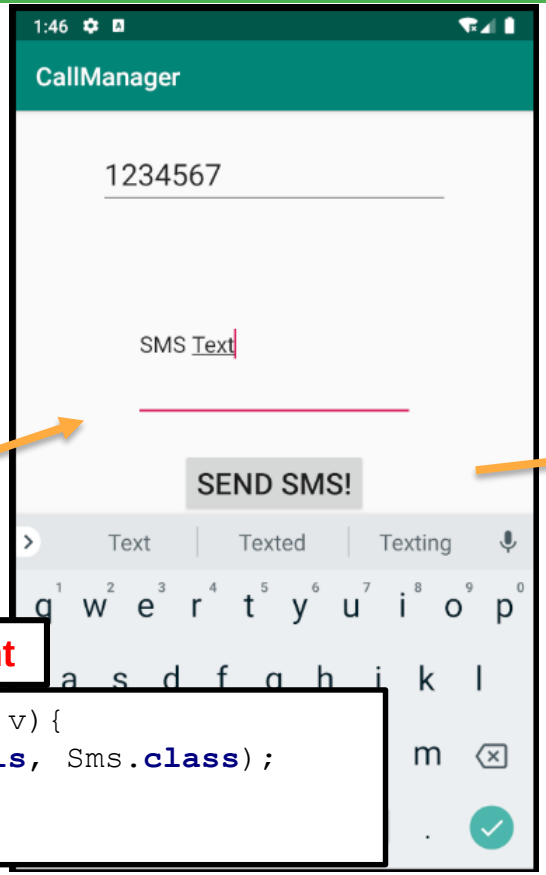


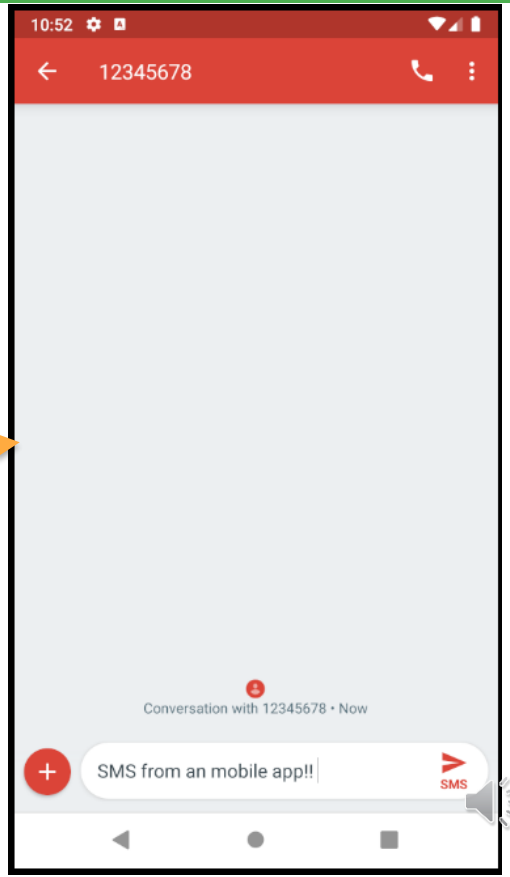
Explicit Intent

```
public void startCall(View v){
    Intent i=new Intent(this, Call.class);
    startActivity(i);
}
```

Calls and SMS Manager Exercise



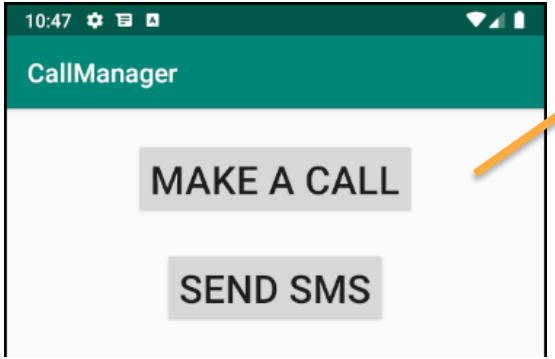


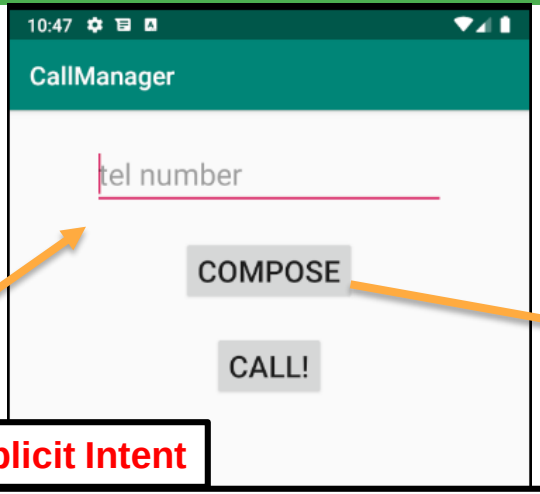


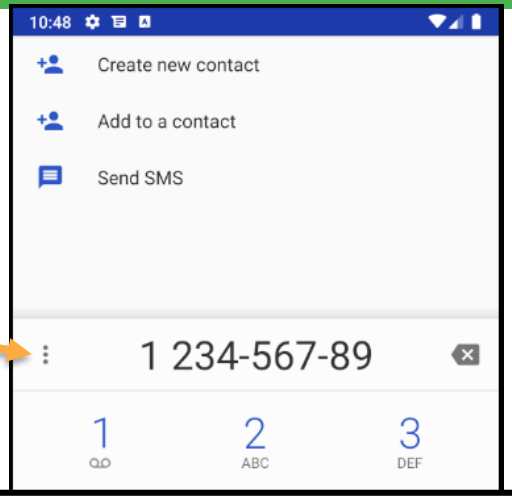
Explicit Intent

```
public void startCall(View v){
    Intent i=new Intent(this, Sms.class);
    startActivity(i);
}
```

Calls and SMS Manager Exercise







Implicit Intent

```
public void compose (View v)
{
    EditText edn =(EditText)findViewById(R.id.editTextNumber);
    Intent intentImplicit=new Intent(Intent.ACTION_DIAL);
    String uri="tel:"+edn.getText().toString();
    intentImplicit.setData(Uri.parse(uri));
    startActivity(intentImplicit);
}
```

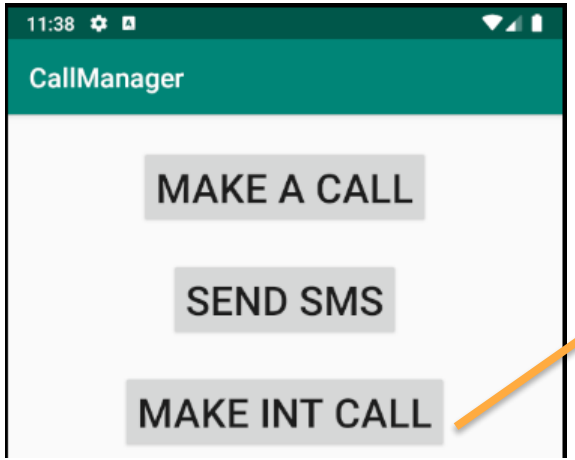
Calls and SMS Manager Exercise

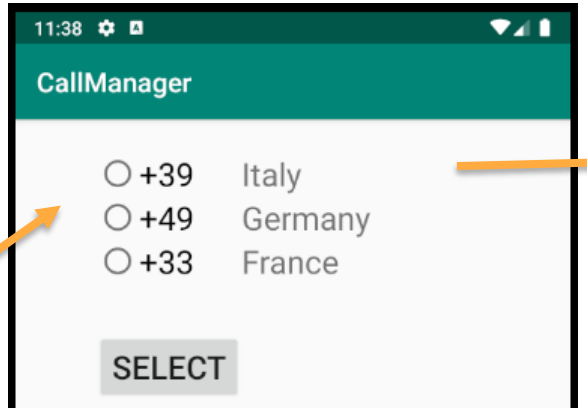
```
public void makeCall (View v)
{
    EditText edn =(EditText)findViewById(R.id.editTextNumber);
    Intent intentImplicit=new Intent(Intent.ACTION_CALL);
    String uri="tel:"+edn.getText().toString();;
    intentImplicit.setData(Uri.parse(uri));
    try {
        startActivity(intentImplicit);
    } catch (SecurityException e) {
        ActivityCompat.requestPermissions(
                Call.this,
                new String[]{Manifest.permission.CALL_PHONE},
                1);
    }
    return;
}
```

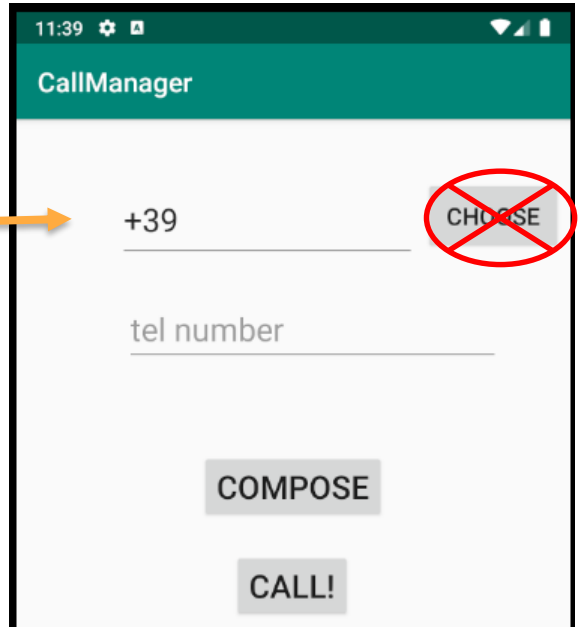
To make a call you need to declare a permission in the manifest....

```
<uses-permission android:name="android.permission.CALL_PHONE" />
```

Calls and SMS Manager Exercise (V.2)

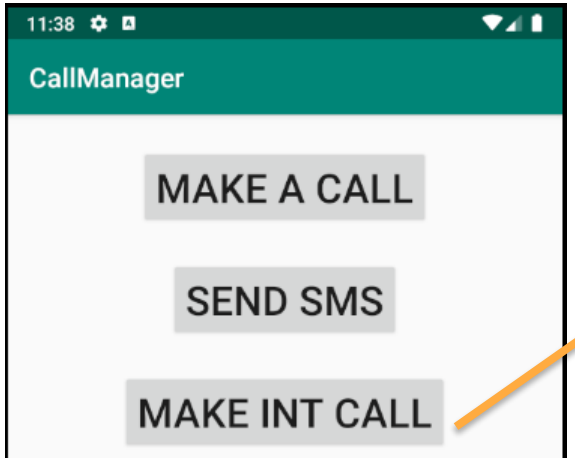


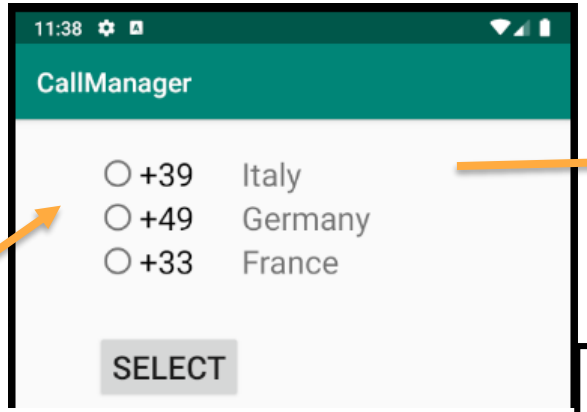


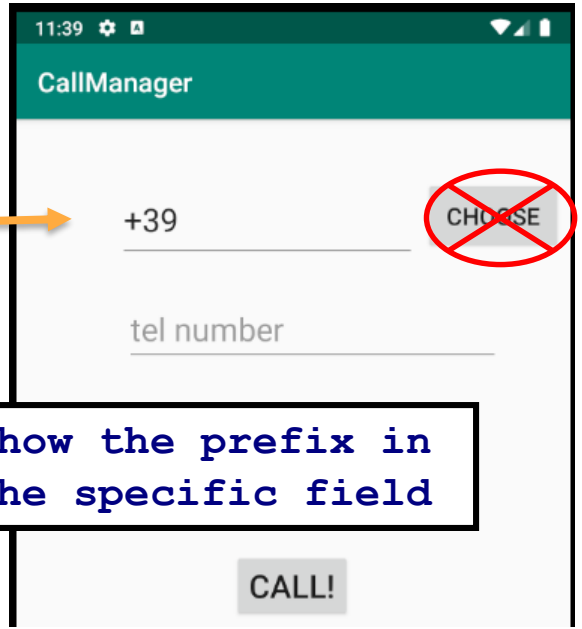



```
Send the selected
Prefix to the next
activity
```

Calls and SMS Manager Exercise (V.2)

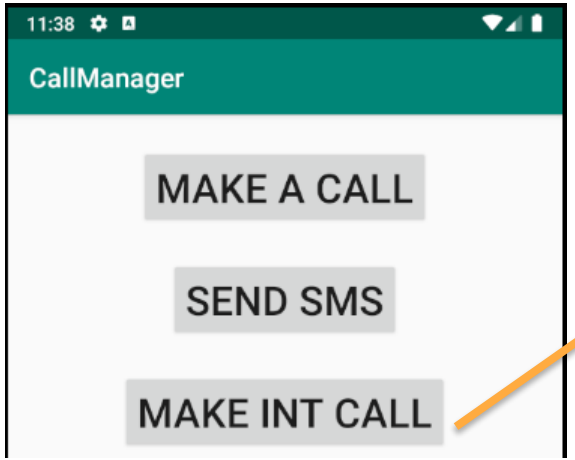


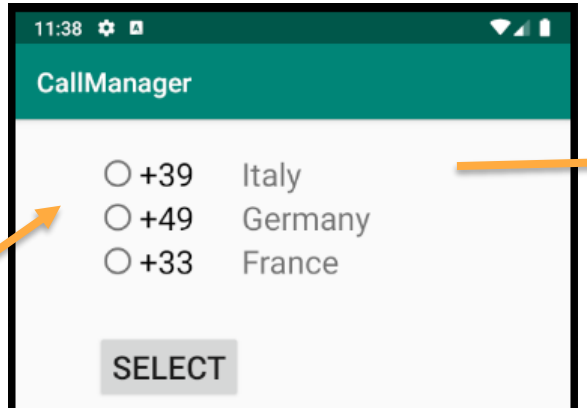


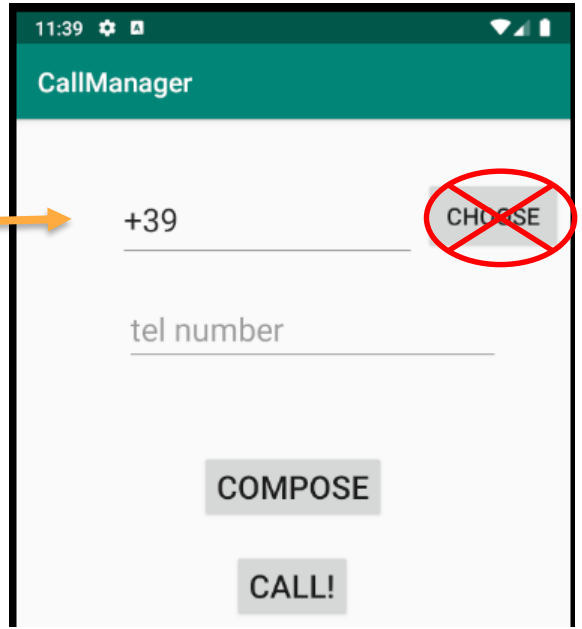


Show the prefix in the specific field

Calls and SMS Manager Exercise (V.2)








```
The prefix is NOT an URI =>
use Extras
```

Calls and SMS Manager Exercise (V.2)

```
public void selectPrefix(View arg0) {
    Intent intent = new Intent(this, CallInternational.class);
    String prefix;
     …
    [ to do ]
     …
    intent.putExtra("prefix", prefix);
    startActivity(intent);
}
```

In the prefix activity

```
@Override
protected void onCreate(Bundle savedInstanceState) {
     …
    Intent intent = getIntent();
    String prefix = intent.getStringExtra("prefix");
    TextView prefixText  = findViewById(R.id.editTextPrefix);
    prefixText.setText(prefix);

}
```

In the activity that allows to compose international numbers