

CS 473 - MDP

Mobile Device Programming

© 2021 Maharishi International University

All course materials are copyright protected by international copyright laws and remain the property of the Maharishi International University. The materials are accessible only for the personal use of students enrolled in this course and only for the duration of the course. Any copying and distributing are not allowed and subject to legal action.



Maharishi International
University

CS 473 - MDP

Mobile Device Programming

MS.CS Program
Department of Computer Science
Renuka Mohanraj , Ph.D.



Maharishi International
University

CS 473 – MDP

Mobile Device Programming

LESSON 5

INTENTS



Maharishi International
University

WHOLENESS

- In this lesson we will examine the usage of intents. An intent is a communication bridge between app component (activity, service or broadcast) via the android system to perform actions. You can also use intents return information to your original activity. *With the help of Transcendental one can communicate easily with inner self like how Android can communicate from one activity to another activity to perform actions with the help of Intents.*

AGENDA

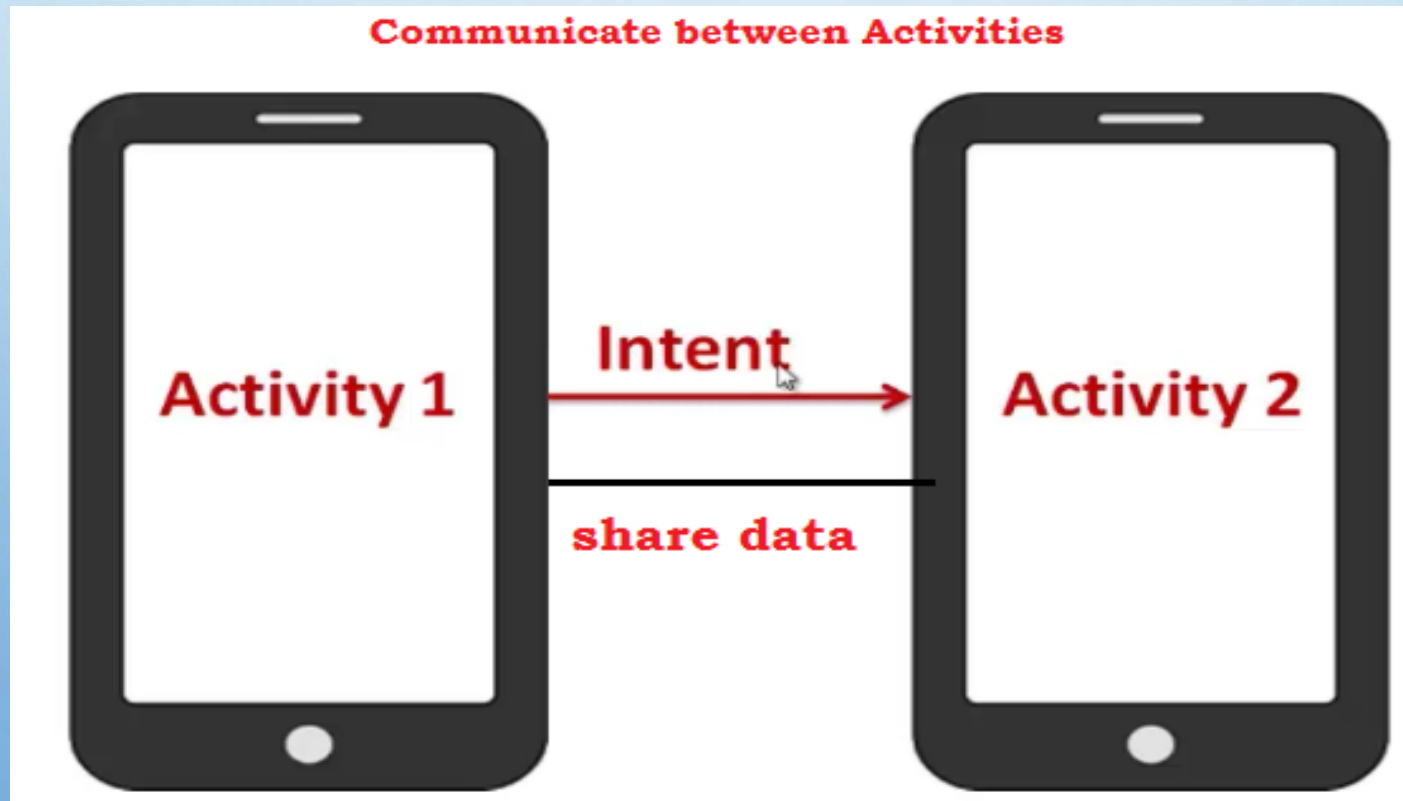
- Activities
 - Implicit intents
 - Explicit intents
 - Hands on example
 - Explicit event - send a message from one activity to another activity.
 - Implicit event – sending message through e-mail, dial up screen and WhatsApp
- Intent Filters
- Getting result from the activity

ACTIVITY & INTENT

- An Intent is a messaging object used to request an action from another app component (activity, service or broadcast) via the android system.
- An Intent can be used to facilitate communication between components :
 - start an Activity
 - start a Service
 - deliver a Broadcast
- Refer: <https://developer.android.com/guide/components/intents-filters>

Introduction to Intents

- We're going to show you how to build apps with multiple activities, and how you can get your apps talking to each other using intents.
- Intents are multi-purpose communication tools, and the Intent class provides different constructors depending on what you are using the intent to do.



Types of intents

- **Explicit intents[user defined]** specify the component to start by name (the fully-qualified class name). You'll typically use an explicit intent to start a component in your own app, because you know the class name of the activity or service you want to start. For example, you can start a new activity in response to a user action or start a service to download a file in the background.
- **Implicit intents[API]** An implicit intent allows you to start an activity in another app by describing an action you intend to perform, such as "share an article", "view a map", or "take a picture". An implicit intent specifies an action and may provide data with which to perform the action. Implicit intents do not specify the target activity class, just the intended action.
- Intents are created using **Intent** class. Following are the major methods in Intents class.

setAction()

setData()

putExtra()

setComponent()

setType()

getExtra()

Hands on Example 1 – Explicit Intent

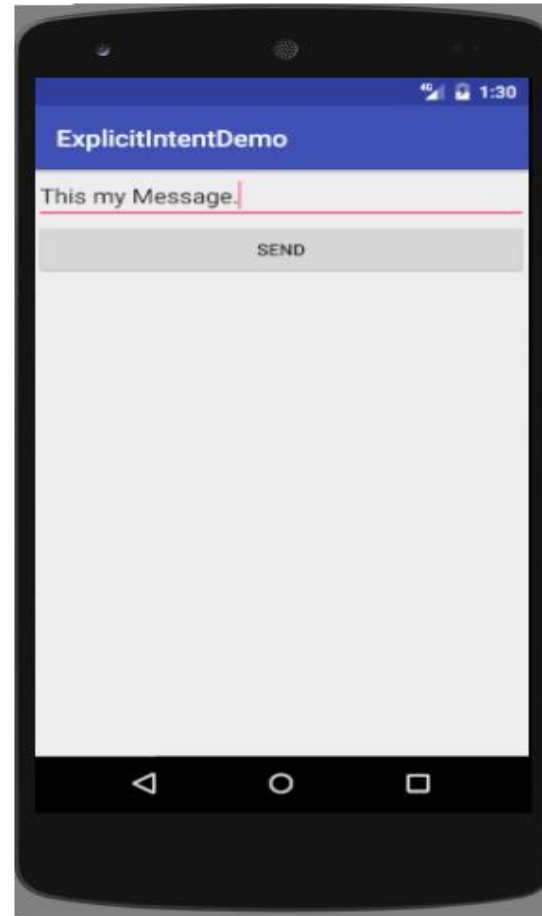
Problem : Send a message from one activity to another activity.



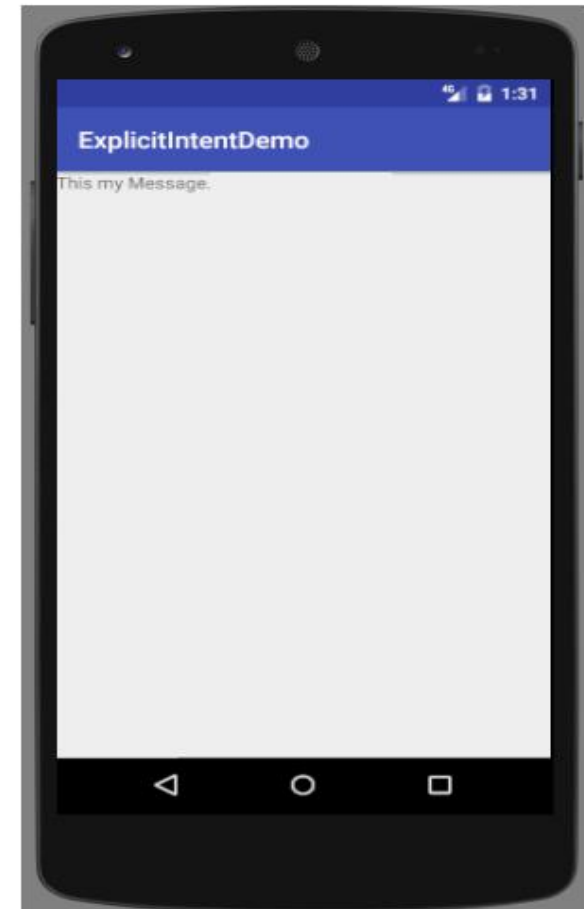
Problem Requirement

Need to Send a message from First Activity to Second Activity using Explicit Intent.

First Activity



Second Activity

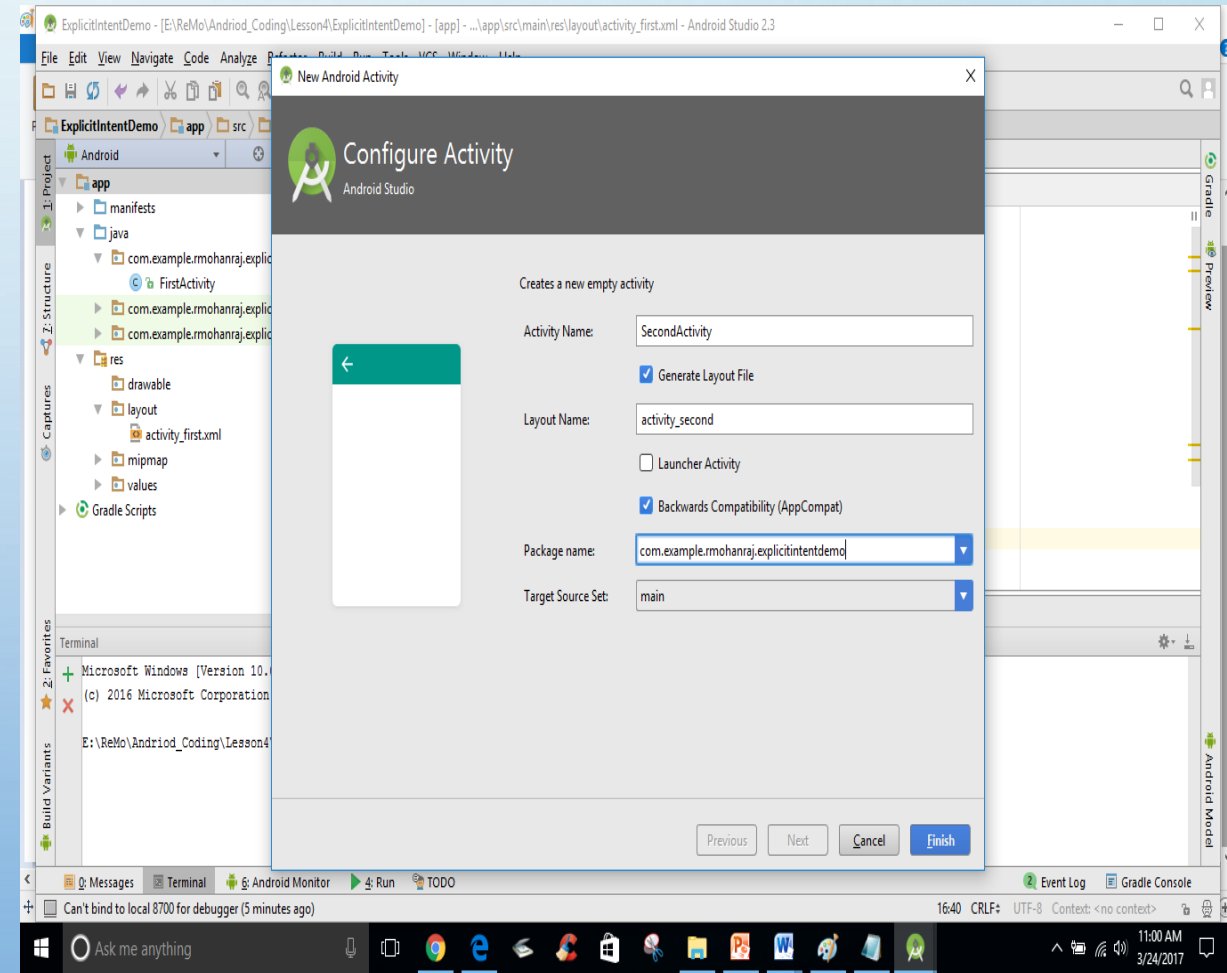
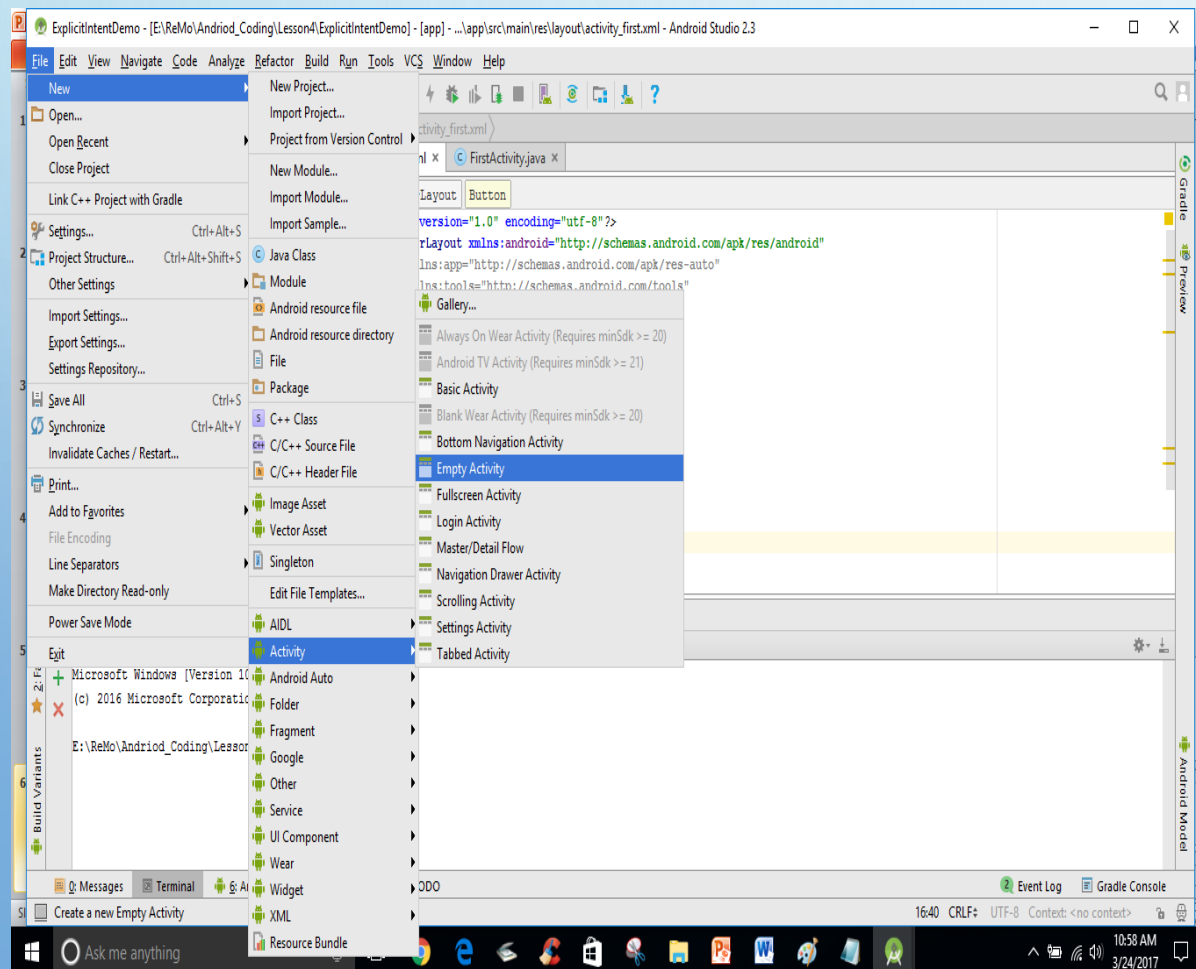


Steps to find the solution

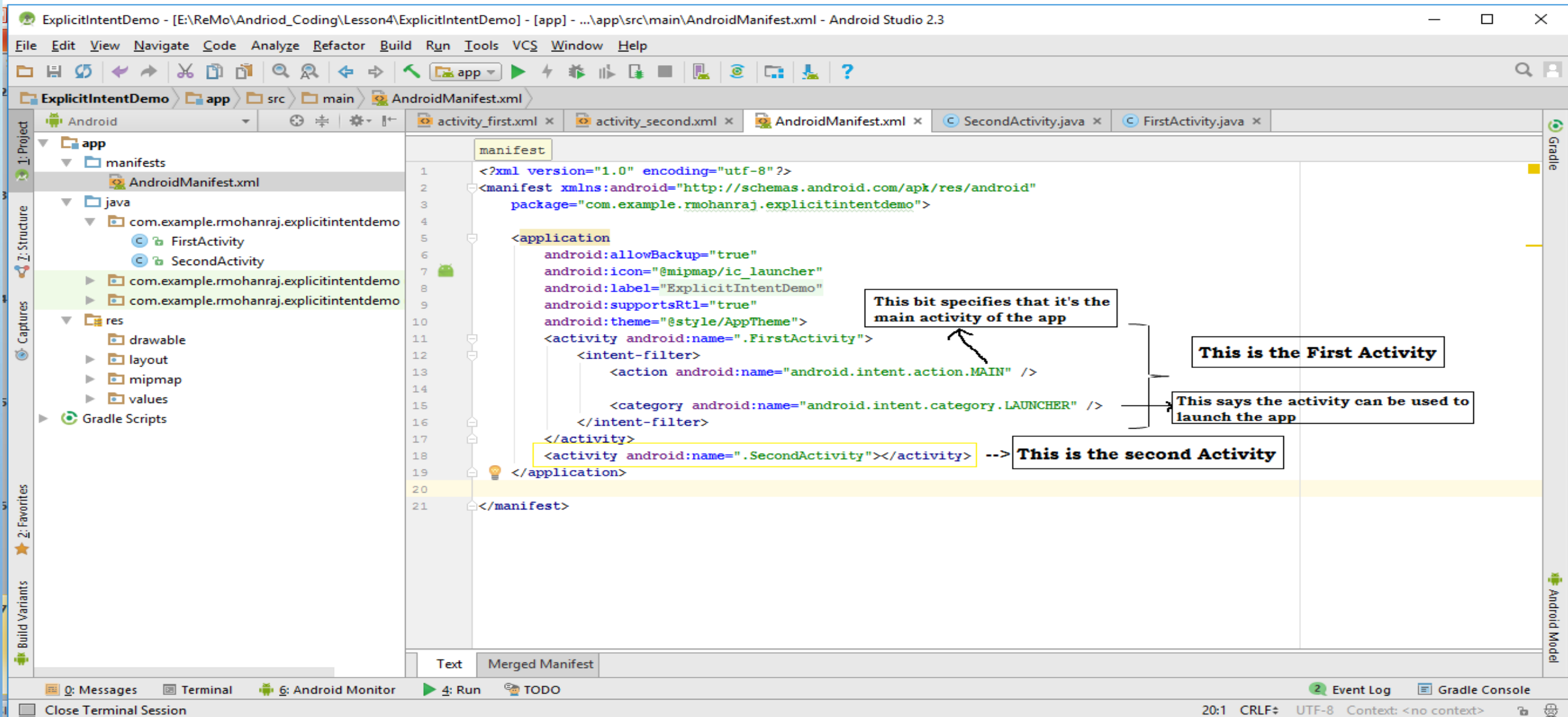
SEND MESSAGE

A little message

- Step 1: Create your first activity with the controls of one text box and one button.
- Step 2 : Create a second activity with one Text view to receive message from the first activity in the same project by choosing File → New → Activity and choose the option for Empty Activity



- Step : 3 If new activity configured automatically at AndroidManifest.xml
`<activity android:name=".SecondActivity"></activity>` as per the screenshot.



- **Step : 4 : Create an Intent and StartActivity**

You can create and send an intent using just a couple of lines of code. You start by creating the intent like this:

```
val intent = Intent(this, SecondActivity::class.java)
```

similar like in java code **Intent intent = new Intent(this,TargetClassname.class);**

The first parameter tells Android which object the intent is from, and you can use the word **this** to refer to the current activity. The second parameter is the class name of the activity that needs to receive the intent. Once you've created the intent object, you pass it to Android like this:

```
startActivity(intent);
```

This tells Android to start the activity specified by the intent. Once Android receives the intent, it checks everything's OK and tells the activity to start. If it can't find the activity, it throws an `ActivityNotFoundException`.

```
fun onSendMessage(view:View){  
    var input = smsg.text.toString()  
    val intent = Intent(this, SecondActivity::class.java)  
    startActivity(intent)  
}
```


- **Step 5: Pass text to the Second Activity from the First Activity.**

- Put the data into the intent as Key/Value pairs. To do this, you use the `putExtra()` method

- **`intent.putExtra("message", value);`**

- Change the `onSendMessage()` with `putExtra()`.

```
fun onSendMessage(view:View){  
    var input = msg.text.toString()  
    val intent = Intent(this, SecondActivity::class.java)  
    intent.putExtra("message",input ) // Here message is a key to retrieve the input text from the  
                                     //second activity  
    startActivity(intent)  
}
```

The `putExtra()` method is overloaded so value has many possible types. As an example, it can be a primitive such as a boolean or int, an array of primitives, or a String. You can use `putExtra()` repeatedly to add numerous extra data to the intent. If you do this, make sure you give each one a unique name.

• **STEP 6 : Retrieve extra information from an intent in the Second Activity**

- There are a couple of useful methods that can help with this. The first of these is **getIntent()**;
- **getIntent()** returns the intent that started the activity, and you can use this to retrieve any extra information that was sent along with it.
- How you do this depends on the type of information that was sent.
- As an example, if you know the intent includes a string value with a name of “message”, you would use the following:

```
val rintent = getIntent() // val rintent = intent
```

```
val output = intent.getStringExtra("message")
```

You're not just limited to retrieving String values. As an example, you can use

```
int intNum = intent.getIntExtra("name", default_value);
```

to retrieve an int with a name. `default_value` specifies what int value you should use as a default.

- **Write this code in the second activity class.**

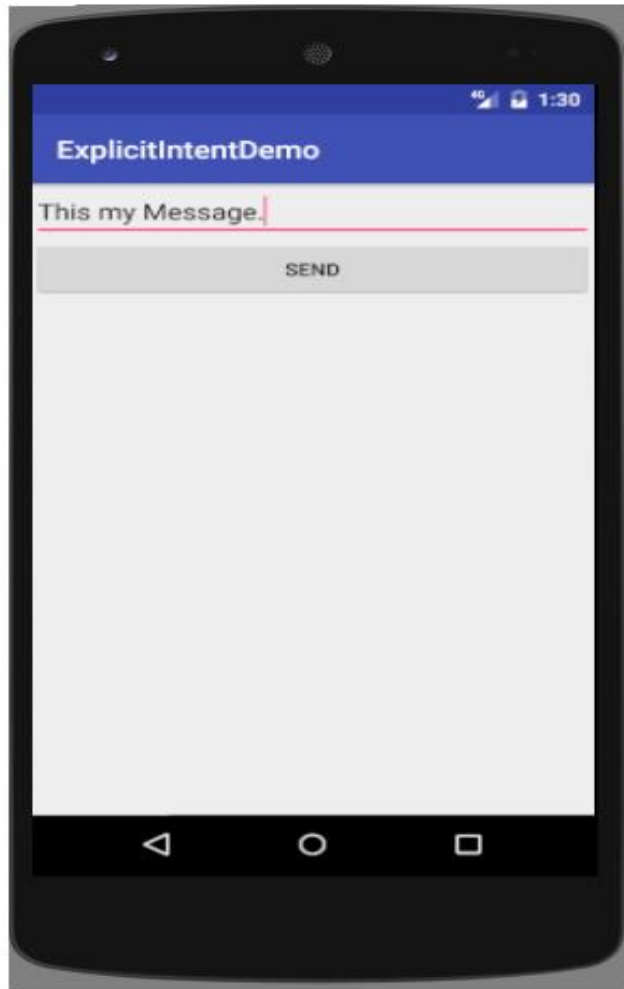
```
class SecondActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_second)  
        val intent = getIntent()  
        val output = intent.getStringExtra("message")  
        rmsg.text = output  
    }  
}
```

Refer: ExplicitIntentDemo, EmplicitIntentObjectDemo (Share a UserAcocunt Object)

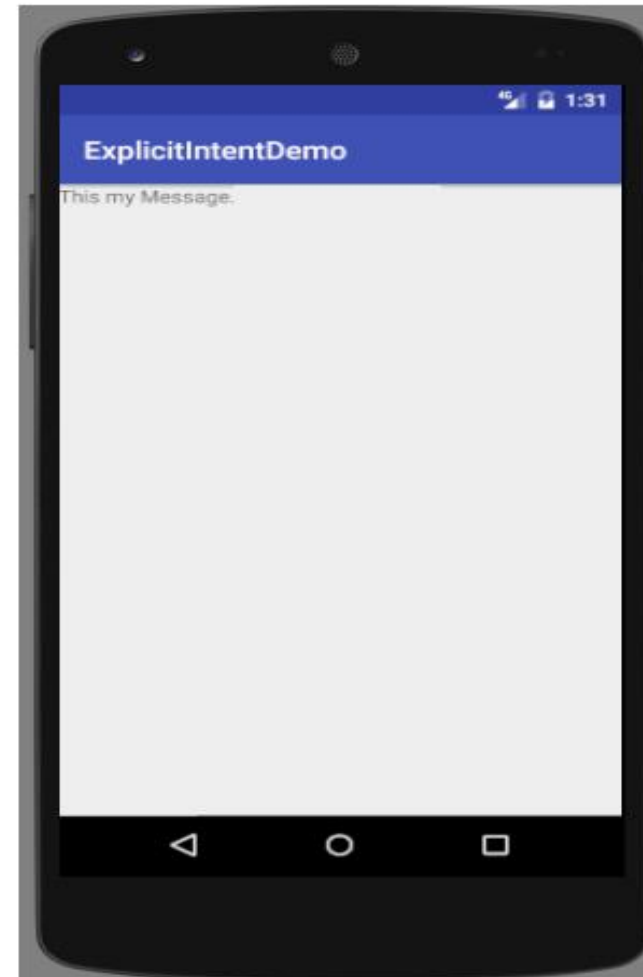
Refer: Updated Explicit Intents by passing message and objects together ExplicitIntentAll

- Step 7 : Run your code. After running the app, you will get result as

First Activity



Second Activity

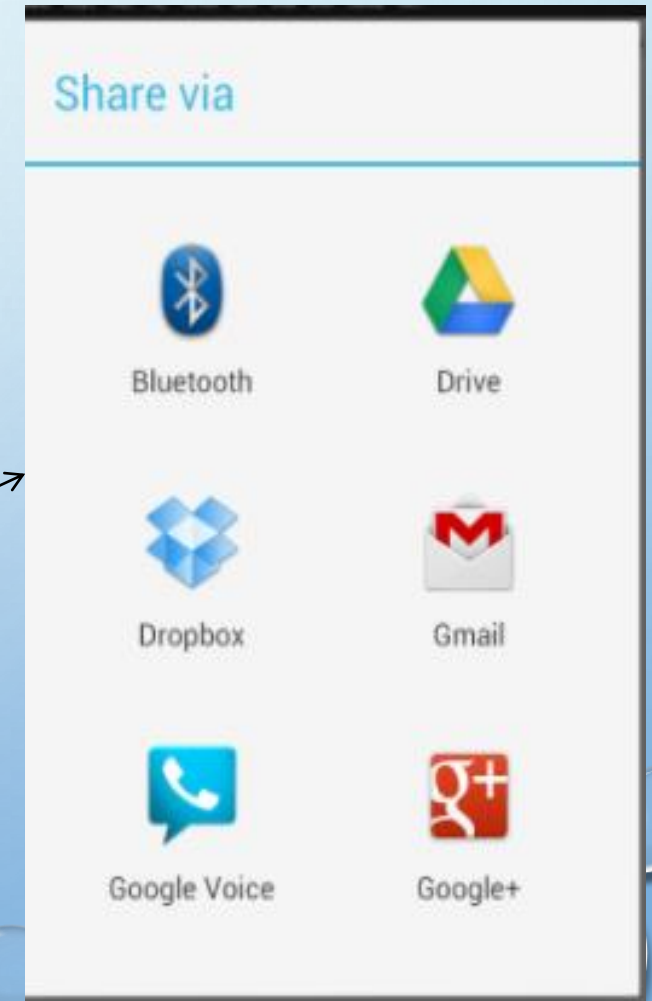


Main Point 1

- While creating an Intent object when we explicitly specify and pass on the target component name directly in the intent, it's an explicit intent. **Science of Consciousness:** *“Wise in the skill of action are those who explicitly first pull the arrow back before they proceed to shoot it ahead. As the mind becomes established in their own consciousness.*

Implicit intents

- Implicit intents do not specify the target activity class, just the intended action
- Android runtime matches the implicit intent request with registered intent handlers
- If there are multiple matches, an app chooser will open to let the user decide
 - When the android runtime finds multiple registered activities that can handle an implicit intent, it displays an **app chooser** to allow the user to select the handler.



Implicit Intents - Examples

Java Code

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);
```

Open a Map

```
// Iowa latitude and longitude  
Uri uri = Uri.parse("geo:41.8780,93.0977");  
Intent it = new Intent(Intent.ACTION_VIEW,uri);  
startActivity(it);
```

Kotlin Code

Show a web page

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

Dial a phone number

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

Open a Map

```
// Iowa latitude and longitude  
val uri = Uri.parse("geo:41.8780,93.0977");  
val it = Intent(Intent.ACTION_VIEW,uri);  
startActivity(it);
```

Avoid exceptions and crashes

- Before starting an implicit activity, use the package manager to check that there is a package with an activity that matches the given criteria.

Example

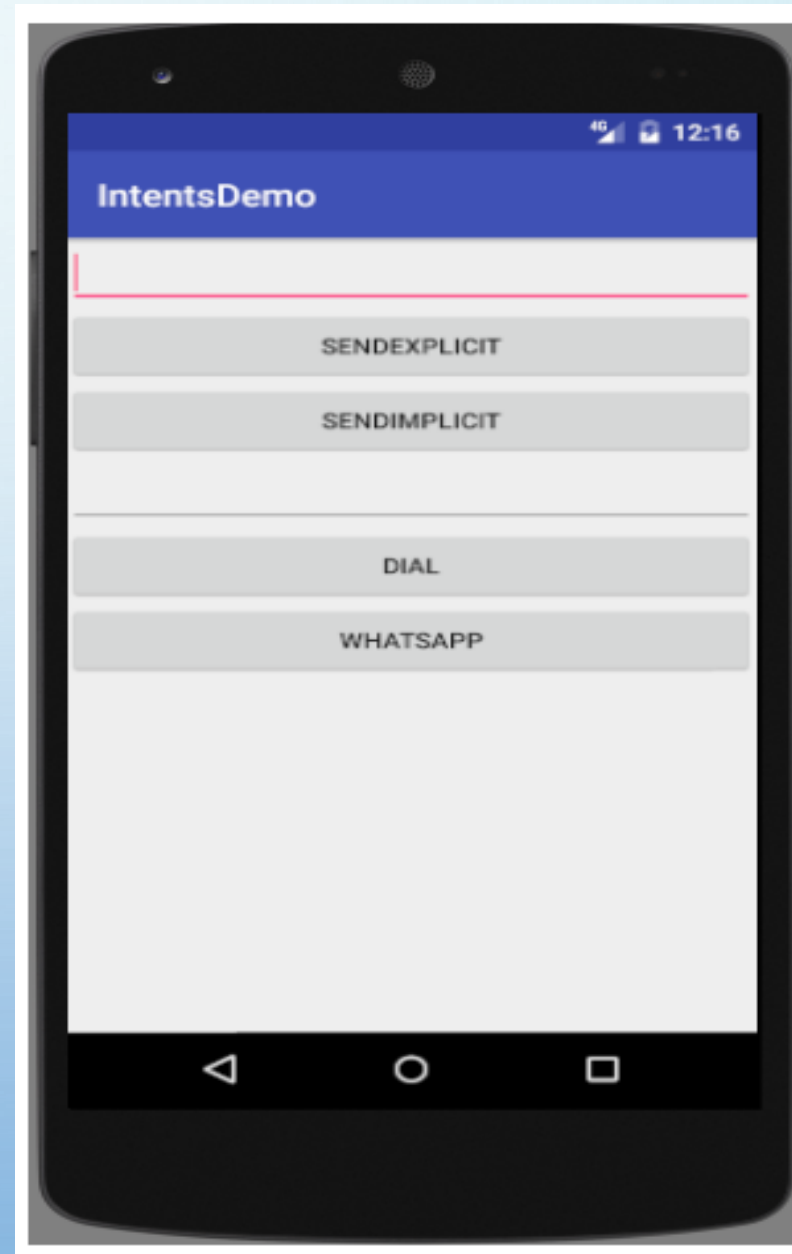
```
Intent intent = new Intent(Intent.ACTION_DIAL);  
intent.setData(Uri.parse("tel:8005551234"));  
if (intent.resolveActivity(getPackageManager()) != null) {  
    startActivity(intent);  
}
```

Hands on Example 2 – Implicit Intent

- However, you can also start up activities from the API or third-party apps. In those cases, though, you will not have specific Intent object representing the other activity in your project, so you cannot use the Intent constructor to initiate.
- Instead, you will use what are referred as the “Implicit” Intent structure.
- Let's enhance the previous example by introducing implicit intents to perform the following.
 - Sending message to the person through Email
 - Showing a Dial screen App (**INTENT.ACTION_DIAL**)
 - Showing a WhatsApp App

DESIGN SCREEN

Refer : IntentsDemo
Shows both Explicit and
Implicit



IMPLICIT INTENT TO SEND MESSAGE

XML Code for the Button SendImplicit

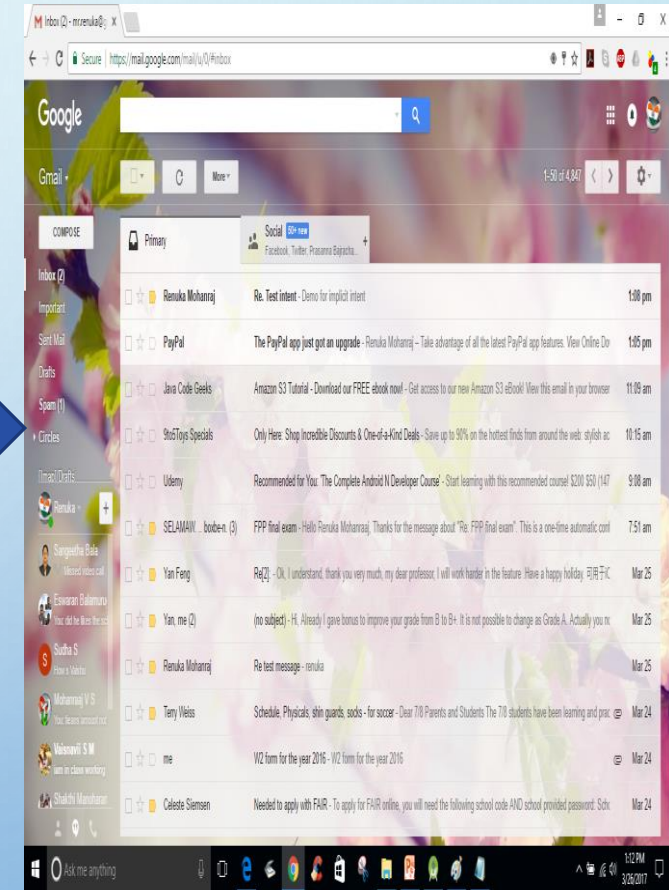
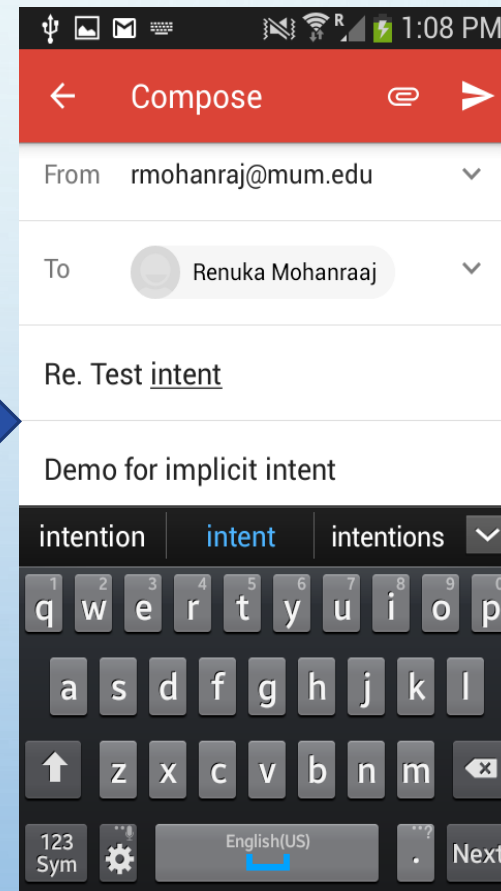
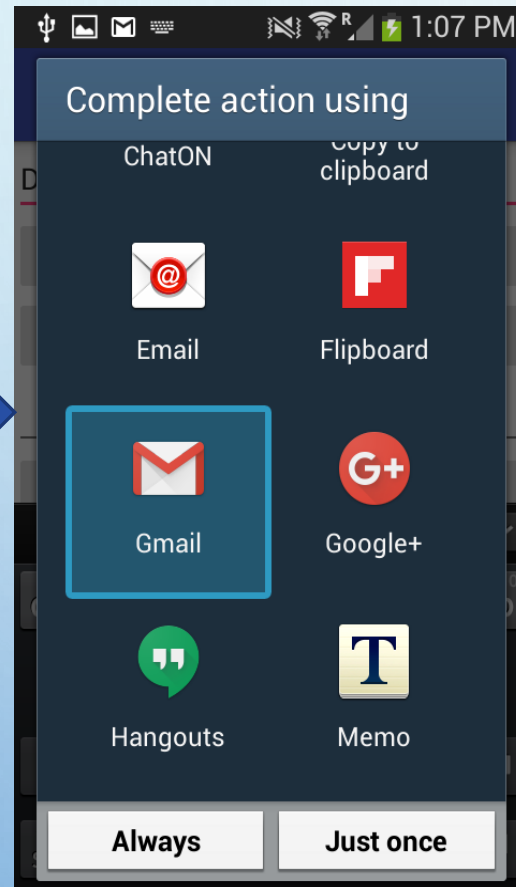
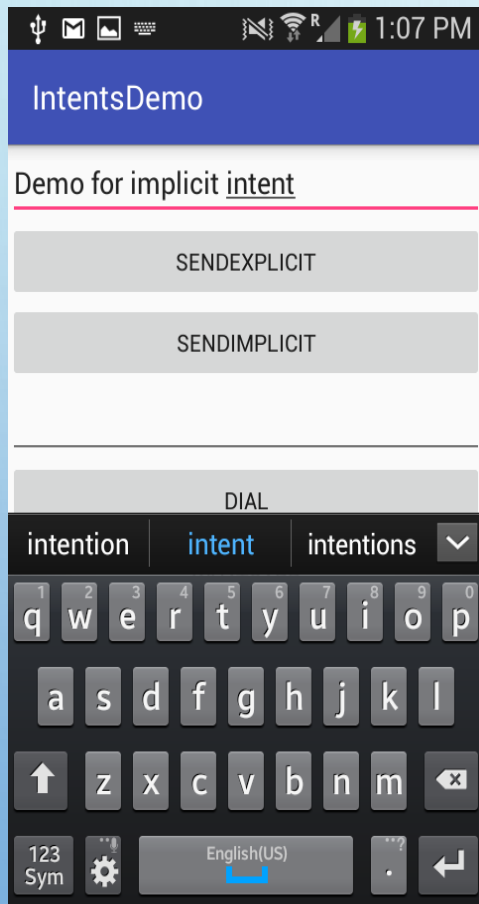
```
<Button  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:id="@+id/send2"  
  
    android:onClick="onSendMessageImplicit"  
    android:text="SendImplicit"/>
```

Kotlin Code for the Button Click Event

```
fun onSendMessageImplicit(view: View) {  
    var input = msg.text.toString()  
    val intent = Intent()  
    intent.action = Intent.ACTION_SEND  
    intent.type = "text/plain"  
    intent.putExtra(Intent.EXTRA_TEXT, input)  
    startActivity(intent)  
}
```

The intent specifies an action of ACTION_SEND, and a MIME type of text/plain.

After clicking **SENDIMPLICIT** button



IMPLICIT INTENT – DIAL SCREEN

<EditText

```
    android:id="@+id/tel"
    android:inputType="phone"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

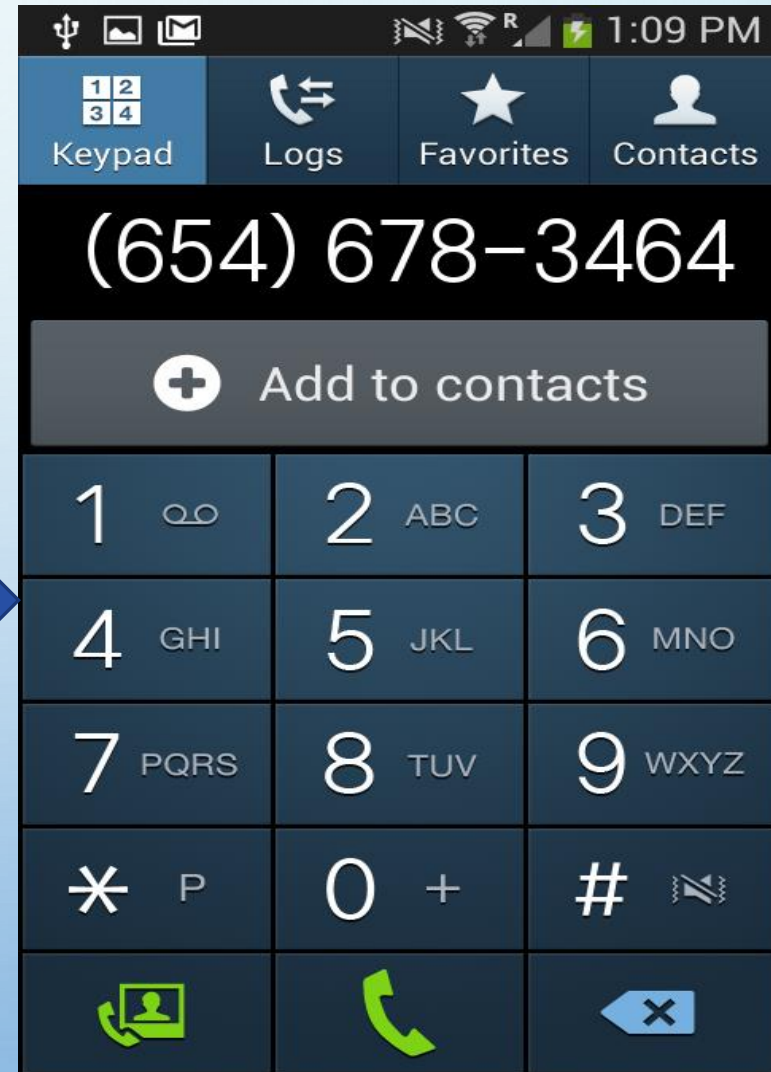
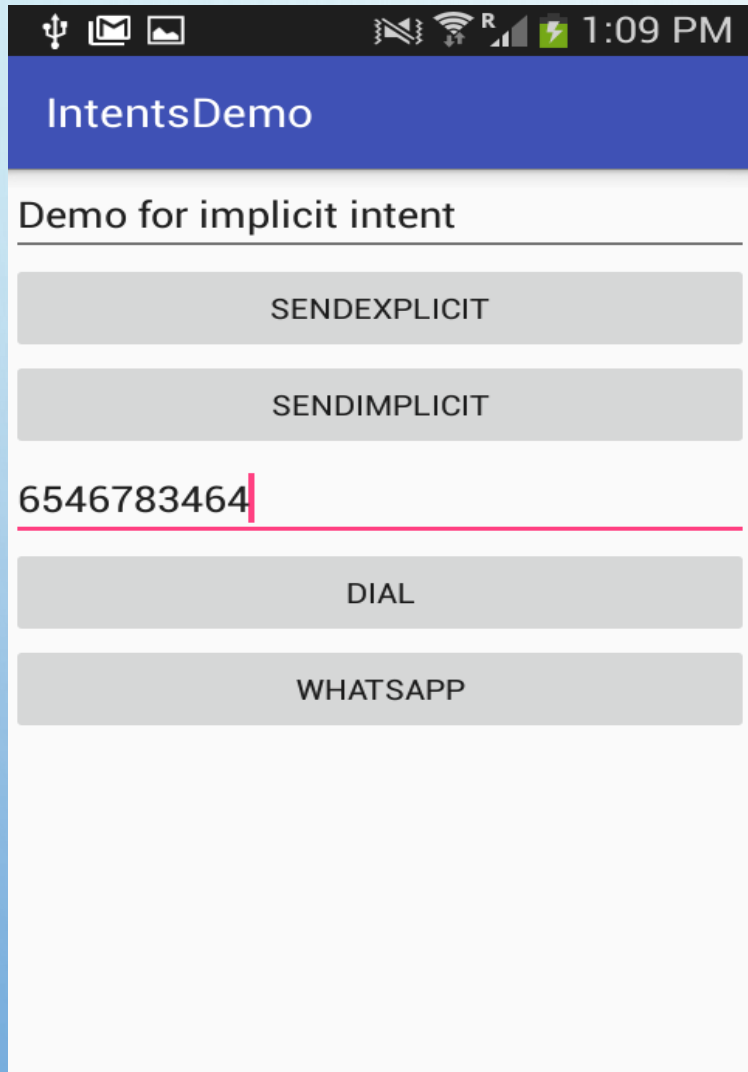
<Button

```
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/dl"
    android:onClick="dial"
    android:text="Dial"/>
```

```
fun dial(view: View) {
    val i = Intent()
    i.action = Intent.ACTION_DIAL
    val et2 = tel.text.toString()
    i.data = Uri.parse("tel:" + et2)
    startActivity(i)
}
```

- Whenever the user clicks the DIAL button, calling built-in dial screen by invoking Intent.ACTION_DIAL to call number when the application is launched.
- To send any data to the built-in activity use setData(), the parameter for this method is Uri component. To get the Uri component use Uri.parse() method which takes the String input along with what type of data we are sending. Here we are sending telephony type of data which accepts number, #, and *. Ordinary number accepts 0-9.
- tel : A data representing kind of telephone number.

After clicking **DIAL** button



IMPLICIT INTENT-WHATSAPP

<Button

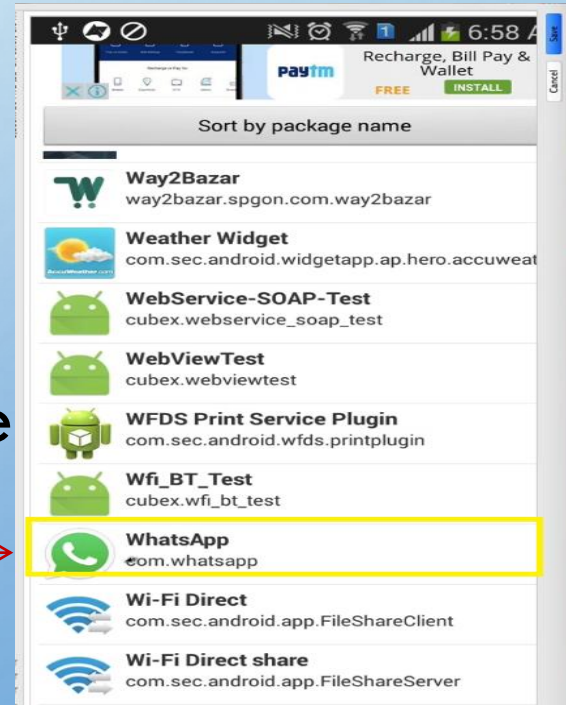
```
android:layout_width="match_parent"  
android:layout_height="wrap_content"  
android:onClick="whatsapp"  
android:text="WhatsApp"/>
```

```
fun whatsapp(view: View) {  
    // To open Third Party app Whatsapp directly  
    val i = packageManager.getLaunchIntentForPackage  
        ("com.whatsapp")  
  
    if(i!=null)  
        startActivity(i)  
    else  
        Toast.makeText(applicationContext,"WhatsApp  
not installed",Toast.LENGTH_LONG).show()  
}
```



How to know the installed app package name – Method 1

If you want to call any third party application in your logic, first you should get the package name from your App called as PackageNameViewer. If you don't have the application install the app from play store. Then see package name and use that in the code.

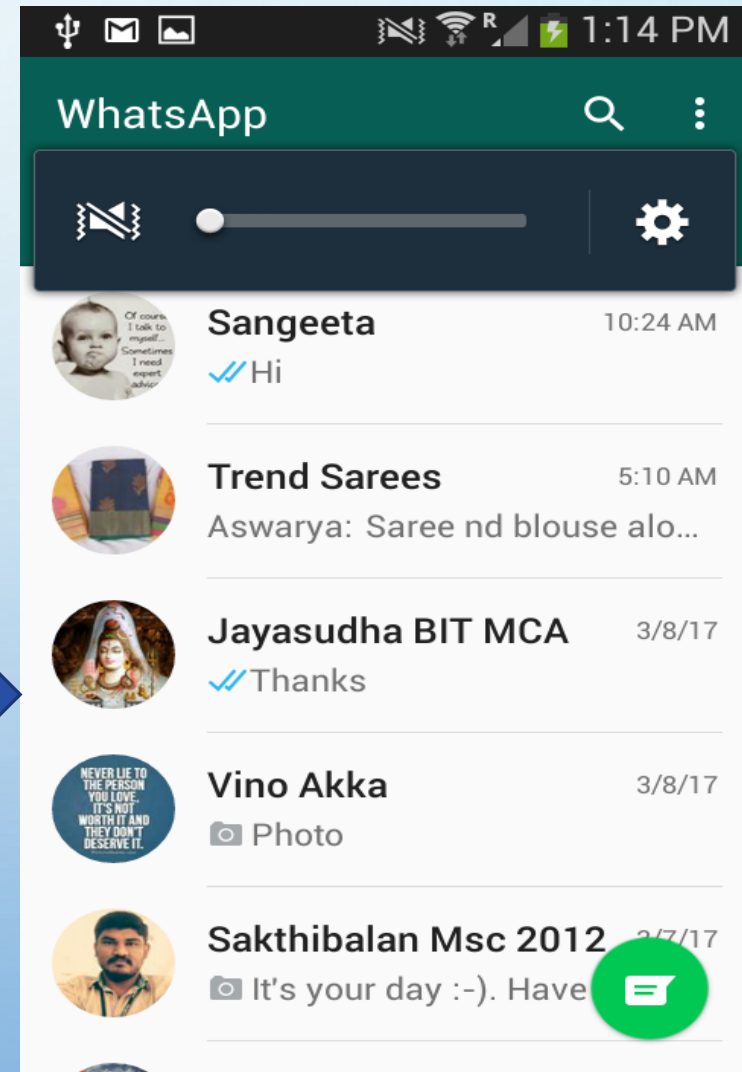
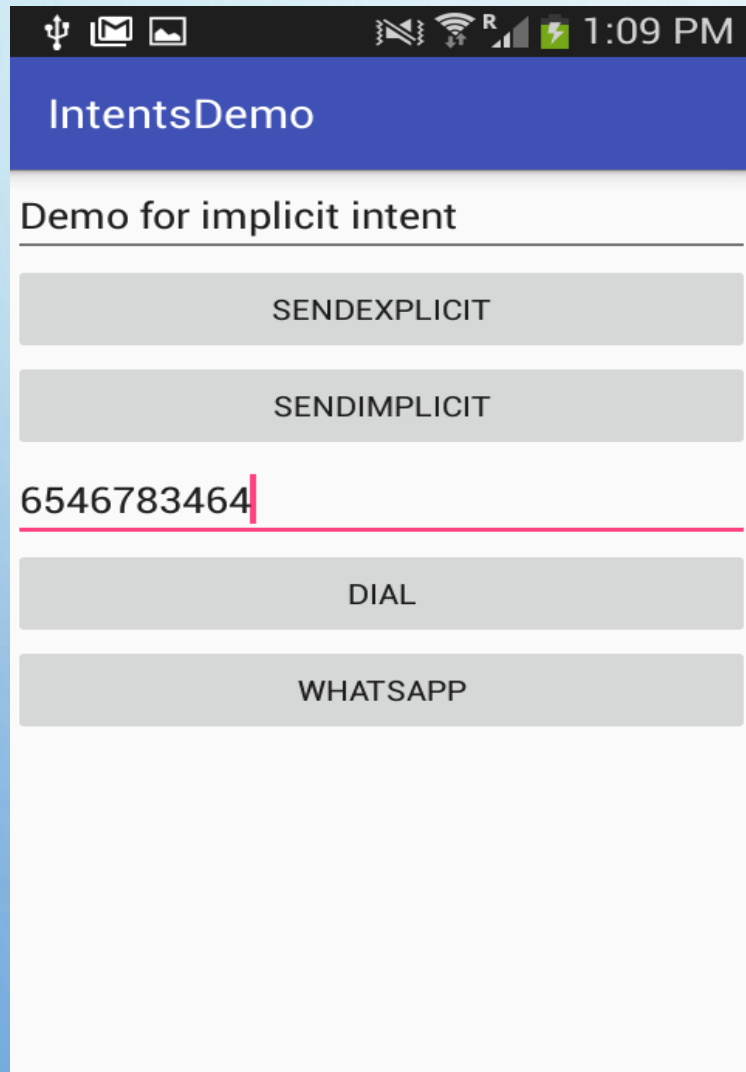


How to know the installed app package name

- **METHOD 2**

- Open [play.Google.Com](https://play.google.com) in your web browser.
- Use the search bar to look for the app for which you need the package name.
- Open the app page and look at the url. The package name forms the end part of the URL i.E. After the *id=?*. Copy it and use it as needed.
- Example <https://play.google.com/store/apps/details?id=com.whatsapp>

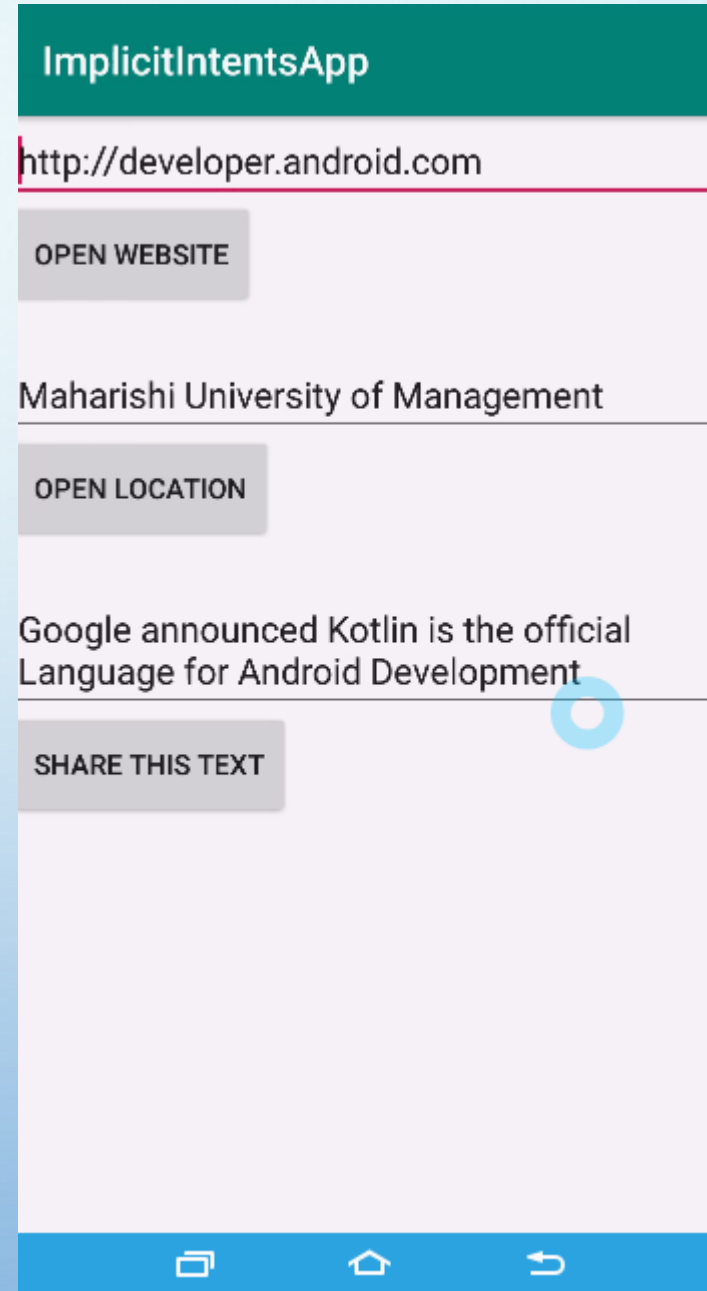
After clicking **WHATSAPP** Button



Hands on Example - 3

ImplicitIntentsUpdated

In this example Share the Text using
ShareCompat.IntentBuilder



Frequently used Implicit Intents

- **Web browser**
- **Email**
- **Alarm clock**
- **Calendar**
- **Camera**
- **Contacts/people app**
- **File storage**
- **Maps**
- **Music or video**
- **Phone**
- **Settings**
- **Text messaging**

Refer for complete information : <https://developer.android.com/guide/components/intents-common.html>

Main Point 2

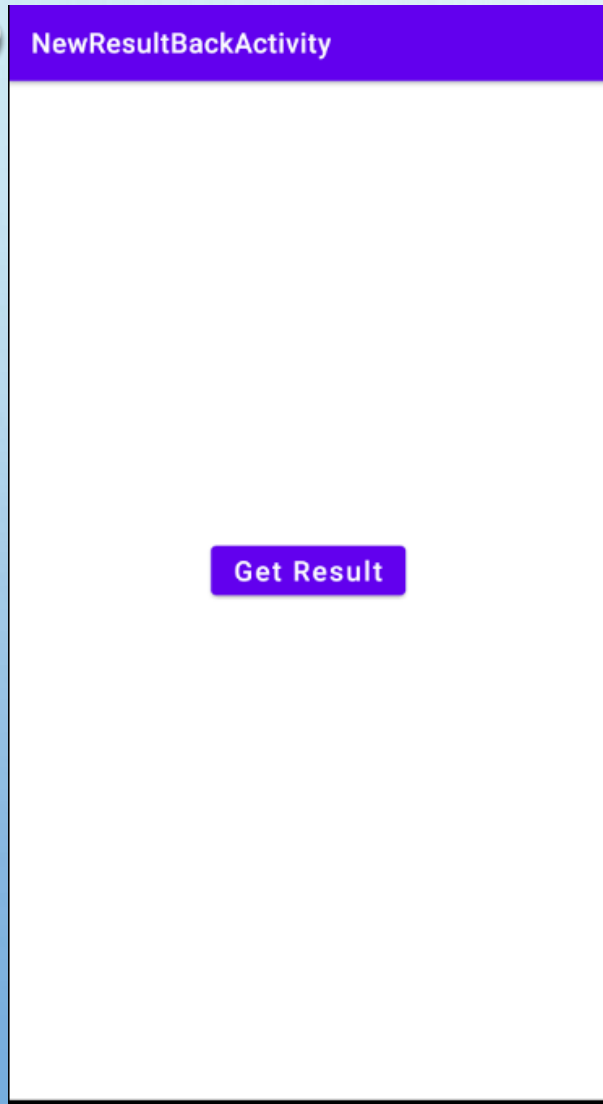
- In implicit intents we delegate the task of evaluating the registered components to Android based on the intent data and the intended action (like send an email, capture a photo, pin location on a map, etc.) that we pass. So, Android will automatically fire up the component from the same app or some other app that can handle the intent message. **Science of Consciousness:** *“Support of Natural Law will render all thought, speech, and action free from stress and strain—life will automatically progress to greater levels of achievement and fulfillment; life will naturally be easy, without problems or failures.*

Getting a result back from an activity

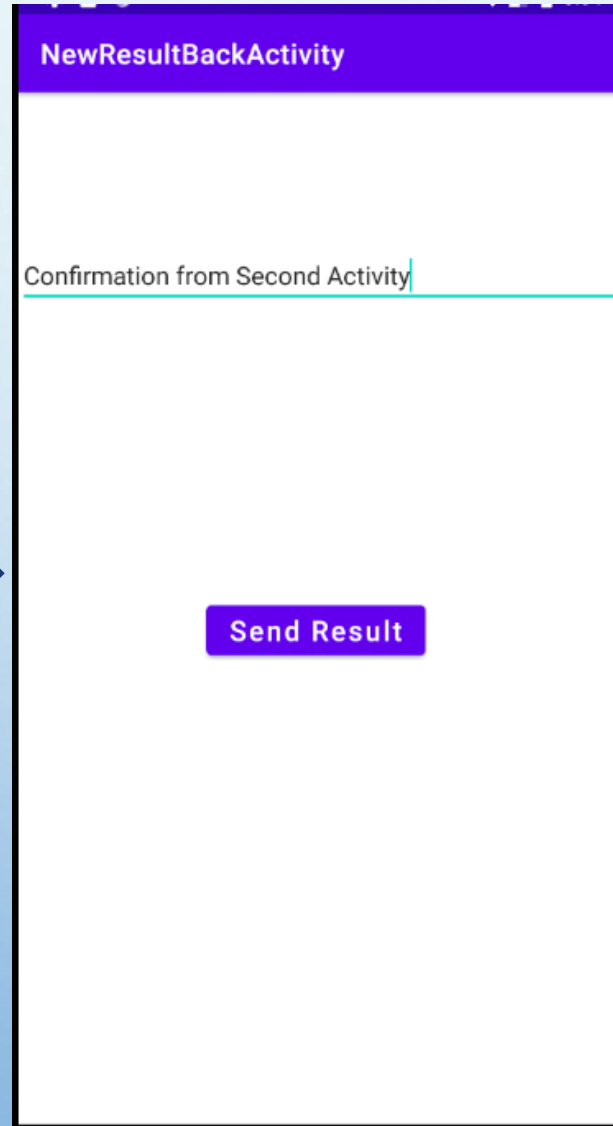
- Starting another activity, whether one within your app or from another app, doesn't need to be a one-way operation. You can also start another activity and receive a result back.
- For example, your app can start a camera app and receive the captured photo as a result. Or you might start the People app for the user to select a contact and you'll receive the contact details as a result.
- The activity result apis provide components for registering for a result, launching the result, and handling the result once it is dispatched by the system.
- It is strongly recommended to use the activity result apis introduced in AndroidX Activity
- A result code specified by the result back activity is either `RESULT_OK` if the operation was successful or `RESULT_CANCELED` if the user backed out or the operation failed for some reason.

Fetching Result from Second Activity to Main Activity

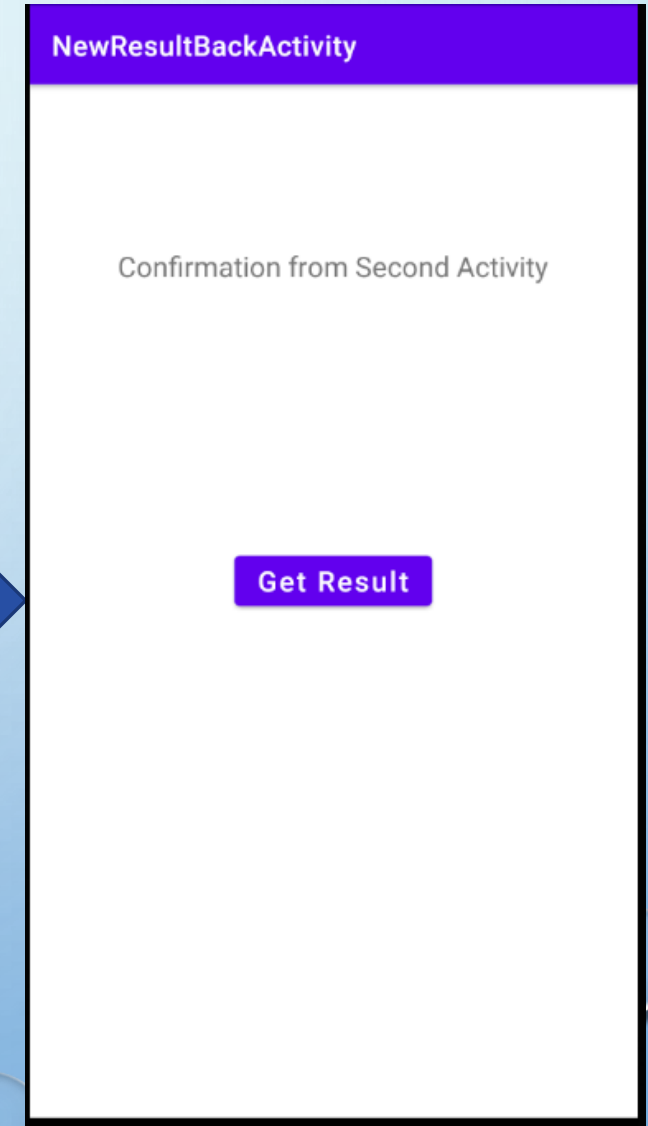
Activity Main



Activity Second



Activity Main



Fetching the result

- In Main activity, write the below code to register for Activity result back.

```
var resultContracts =
```

```
registerForActivityResult(ActivityResultContracts.StartActivityForResult()) {  
result-> // Handle the return result      }
```

Lambda argument result is type of ActivityResult which has the intent, result code such as RESULT_OK or RESULT_CANCEL

registerForActivityResult() takes an ActivityResultContract and an ActivityResultCallback and returns an ActivityResultLauncher which you'll use to launch the other activity.

To Launch: resultContracts.launch(intent object)

Calling launch() starts the process of producing the result.

Getting a result back from other activity

// Partial Coding part from the MainActivity.java

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_main)  
    //Retrieve the Result back doing the below step  
    var resultContracts = registerForActivityResult(ActivityResultContracts.StartActivityForResult()){ result->  
        if(result.resultCode == Activity.RESULT_OK)  
            tv.text = result.data?.data.toString()  
        else  
            tv.text = "Failed to get Result"  
    }  
    btnMain.setOnClickListener {  
        var intent = Intent(this,SecondActivity::class.java)  
        resultContracts.launch(intent)  
    }  
}
```


Getting a result back from a child activity

// Partial Coding part from the SecondActivity.java

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_second)  
    btnSecond.setOnClickListener {  
        var result = etext?.text.toString()  
        val rintent = intent  
        rintent.data = Uri.parse(result)  
        setResult(Activity.RESULT_OK, rintent)  
        finish()  
    }  
}  
  
override fun onBackPressed() {  
    setResult(Activity.RESULT_CANCELED)  
    finish()  
}
```

Note: If you have multiple activity result calls that either use different contracts or want separate callbacks, you can call `registerForActivityResult()` multiple times to register multiple `ActivityResultLauncher` instances. You must always call `registerForActivityResult()`

Demo Code : Find the complete code from `GetBackResultActivity`

Intent Filters

Why do you need Intent Filters

- To allow other apps to start your activity, you need to add an `<intent-filter>` element in your manifest file for the corresponding `<activity>` element.
- For example, if you build a social app that can share messages or photos with the user's friends, it's in your best interest to support the `ACTION_SEND` intent so users can initiate a "share" action from another app and launch your app to perform the action.
- When your app is installed on a device, the system identifies your intent filters and adds the information to an internal catalog of intents supported by all installed apps. When an app calls `startActivity()` or `startActivityForResult()`, with an implicit intent, the system finds which activity (or activities) can respond to the intent.

Intent Filters: Action, Category and Data

Intent Action

The type of thing that the app wants to have done on its behalf.

`android.intent.action.VIEW` — matches all intents with ACTION_VIEW
`android.intent.action.SEND` — matches all intents with ACTION_SEND

Intent Category

Adds a subtype to the action.

category — additional information (list of categories)

`android.intent.category.BROWSABLE`—can be started by web browser
`android.intent.category.LAUNCHER`—Show activity as launcher icon
`android.intent.category.DEFAULT`—In order to receive implicit intents, you must include the `category.DEFAULT` category in the intent filter.

Intent Data Type

MIME Data Type

Allows Activities to support specific data types.

data — Filter on data URIs, MIME type

`android:scheme="https"`—require URIs to be https protocol
`android:host="developer.android.com"`—only accept intents from specified hosts
`android:mimeType="text/plain"`—limit the acceptable types of documents

Registering activity in Manifest

AndroidManifest.xml

```
<activity android:name=".HelloWorld"
    android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.VIEW"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <category android:name="android.intent.category.BROWSABLE"/>
        <
            android:scheme="http" android:host="androidium.org"/>
    </intent-filter>
</activity>
```

HelloWorld.kt

```
val intent = Intent (Intent.ACTION_VIEW,Uri.parse("http://androidium.org"));
startActivity(intent);
```

An activity can have Multiple filters

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

```
  <intent-filter>
```

```
    <action android:name="android.intent.action.SEND"/>
```

```
    ...
```

```
  </intent-filter>
```

```
  <intent-filter>
```

```
    <action android:name="android.intent.action.SEND_MULTIPLE"/>
```

```
    ...
```

```
  </intent-filter>
```

```
</activity>
```


A filter can have multiple actions & data

```
<intent-filter>
```

```
  <action android:name="android.intent.action.SEND_MULTIPLE"/>
```

```
  <category android:name="android.intent.category.DEFAULT"/>
```

```
  <data android:mimeType="image/*"/>
```

```
  <data android:mimeType="video/*"/>
```

```
</intent-filter>
```

Main Point 3

- Intent Filters describe what ACTION, DATA, or CATEGORIES that component can handle. It defined inside your Android Manifest file also dynamically registered in code. It says what Intent can be handled by the event. **Science of Consciousness:** *Through regular practice of TM, Thought leads to action, action leads to achievement in daily activities, and achievement leads to fulfillment.*

UNITY CHART

CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE

Experience of Wholeness on the move

1. Intent is basically an *intention* to do an action. It's a way to communicate between Android components (not just activities) to request an action from and by different components.
 2. Intents activate movement in one direction toward a goal state just as Creative Intelligence is Purposeful and Intelligent, providing one-pointed growth towards the goal.
-
3. **Transcendental Consciousness:** TC is the self-referral field of existence, at the basis of all manifest existence.
 4. ***Impulses within the Transcendental field:*** *These impulses are totally within the Transcendental field, the Transcendent interacting with itself.*
 5. ***Wholeness moving within Itself:*** *In Unity Consciousness, one sees that all activity in the universe springs from the self-referral dynamics of wholeness. The "base case" – the reference point – is always the Self, realized as Brahman.*

