

# Mobile Programming



Activity

# Task and Back Stack (1/4)

## ■ Task

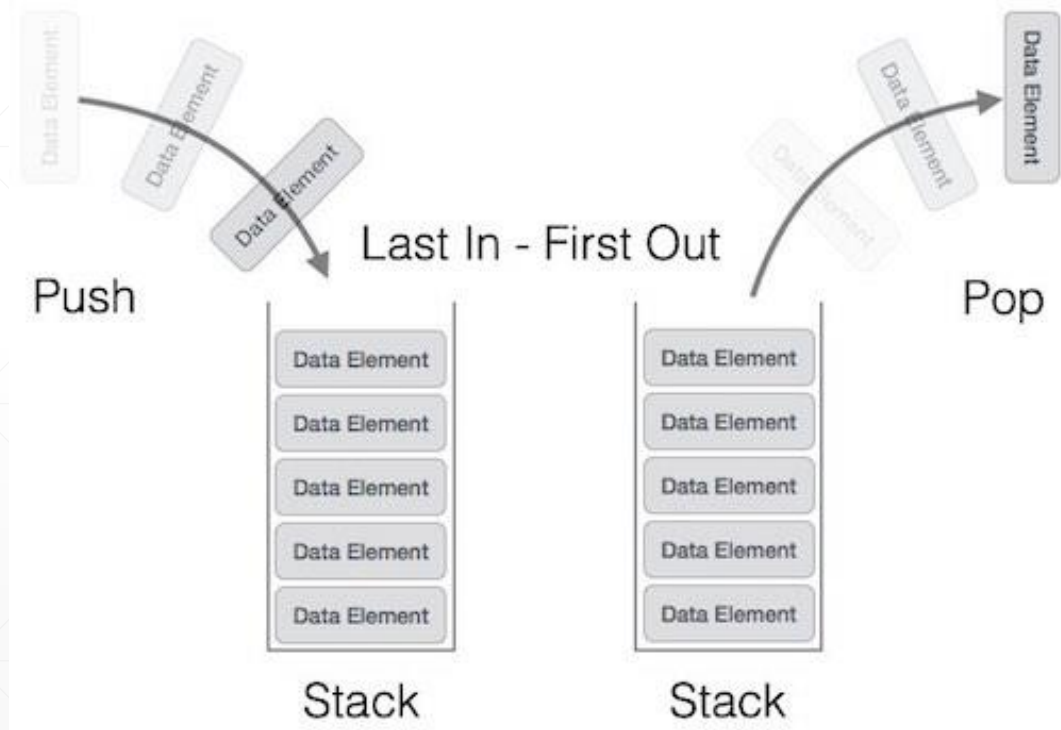
- A collection of activities that users interact with when trying to do something in your app
- If a user launches an app, then a new task is created and the main activity for that app opens as the root activity in the activity/back stack
- Activities of a task are arranged in the stack in the order in which each activity is opened

## ■ Example

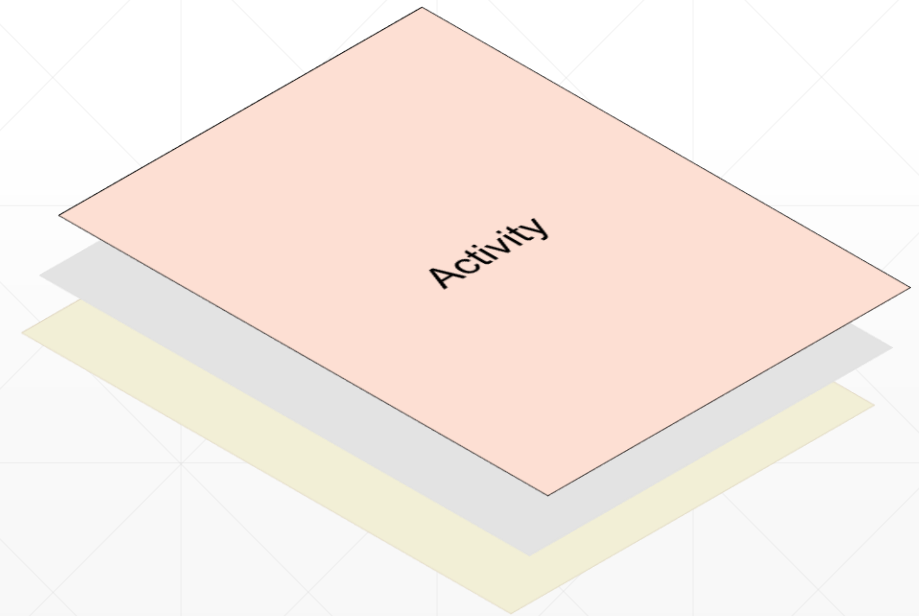
- An email app might have one activity to show a list of new messages
- When the user selects a message, a new activity opens to view that message
  - This new activity is added to the back stack!
- Then, if the user presses or gestures Back, that new activity is finished and popped off the stack

# Task and Back Stack (2/4)

## ■ Activity stack (Back stack)

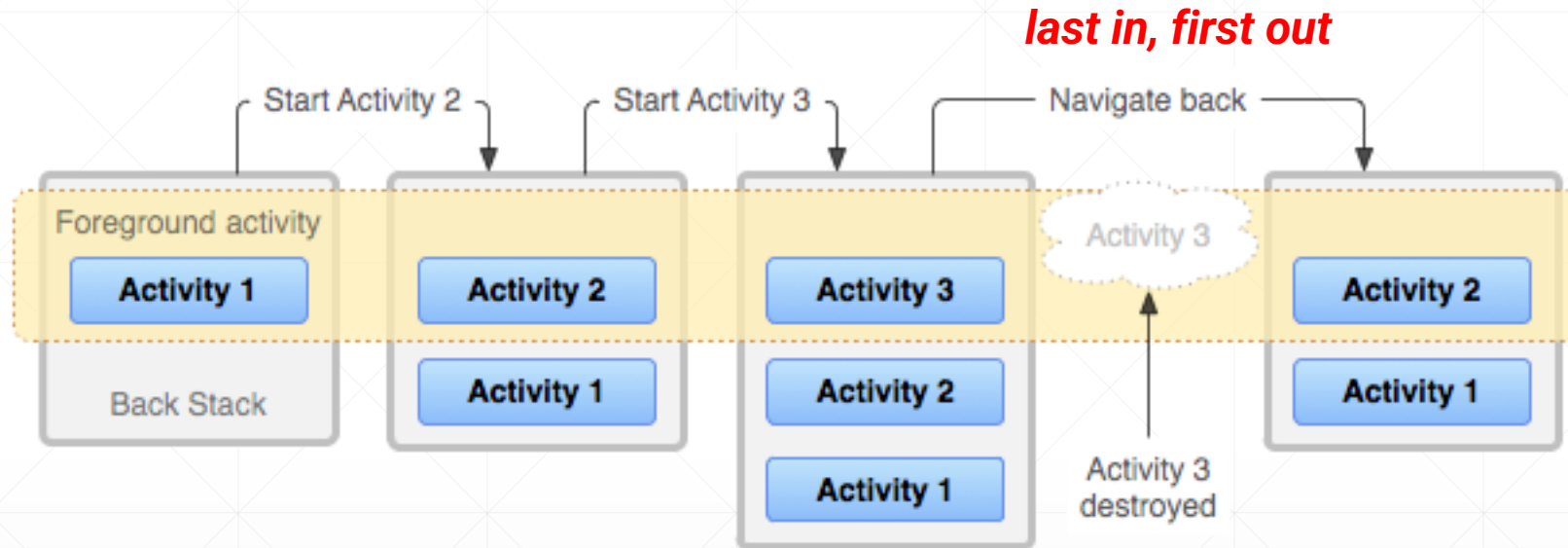


Activity stack



# Task and Back Stack (3/4)

## ■ Activity stack (Back stack)

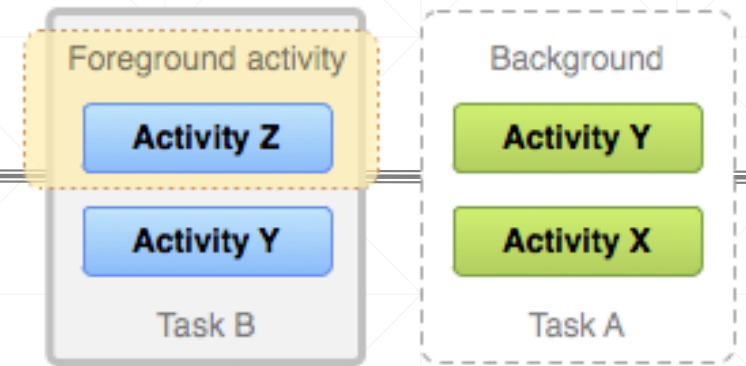


- **Foreground activity**
  - Activity on the top of a stack which interacts with a user
- **Back stack**
  - Activities in the stack are never rearranged, only pushed and popped from the stack
  - Pushed onto the stack when started by the current activity and popped off when the user leaves it using the Back button or gesture

# Task and Back Stack (4/4)

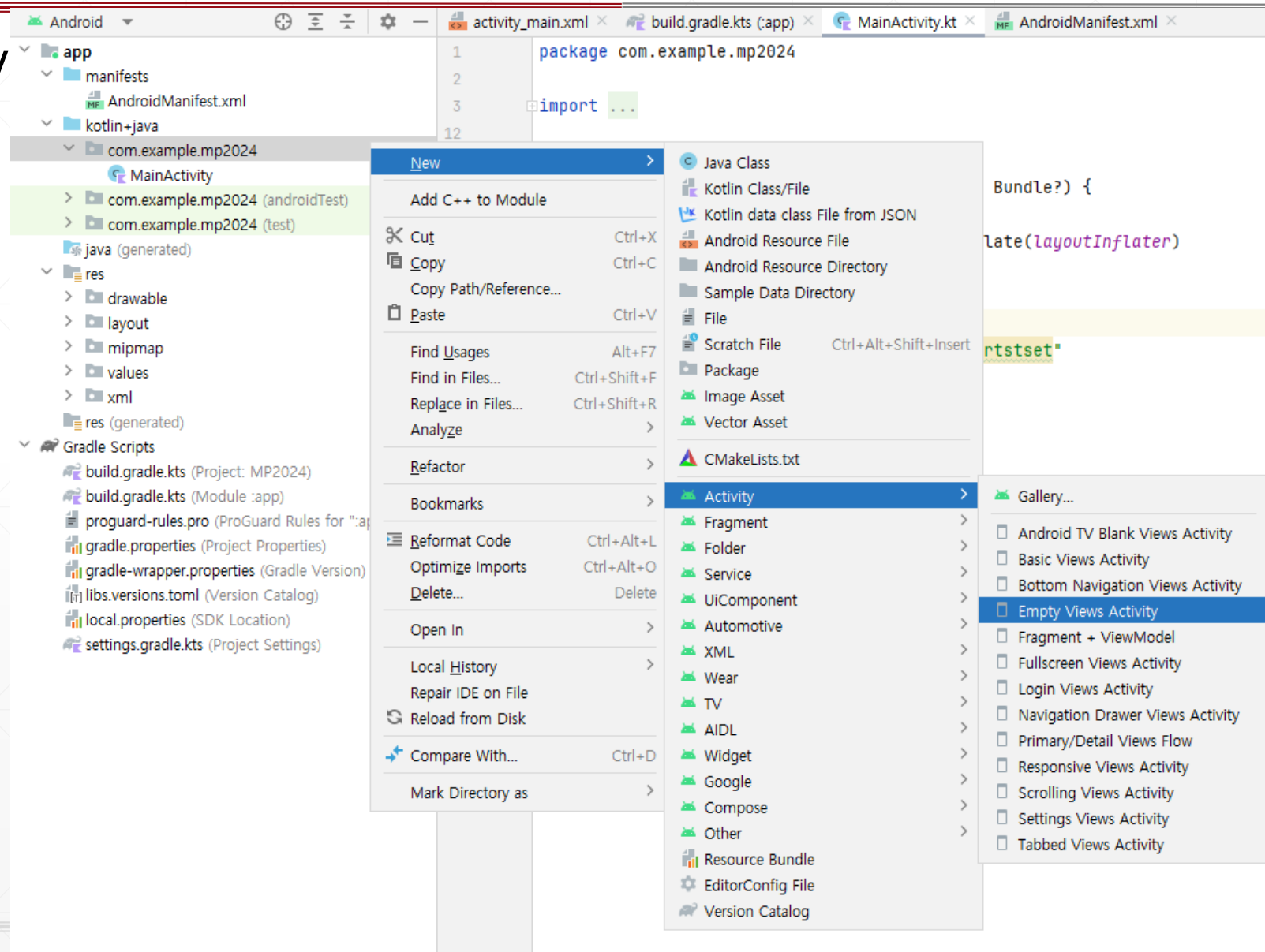
## ■ Foreground and Background tasks

- Task B receives user interaction in the foreground, while Task A is in the background, waiting to be resumed!
- A task is a cohesive unit that can move to the background when a user begins a new task or goes to the Home screen
- Example scenario)
  - When the Home screen appears, Task B goes into the background
  - When a new app (Task A) starts, the system starts a task for that app with its own stack of activities



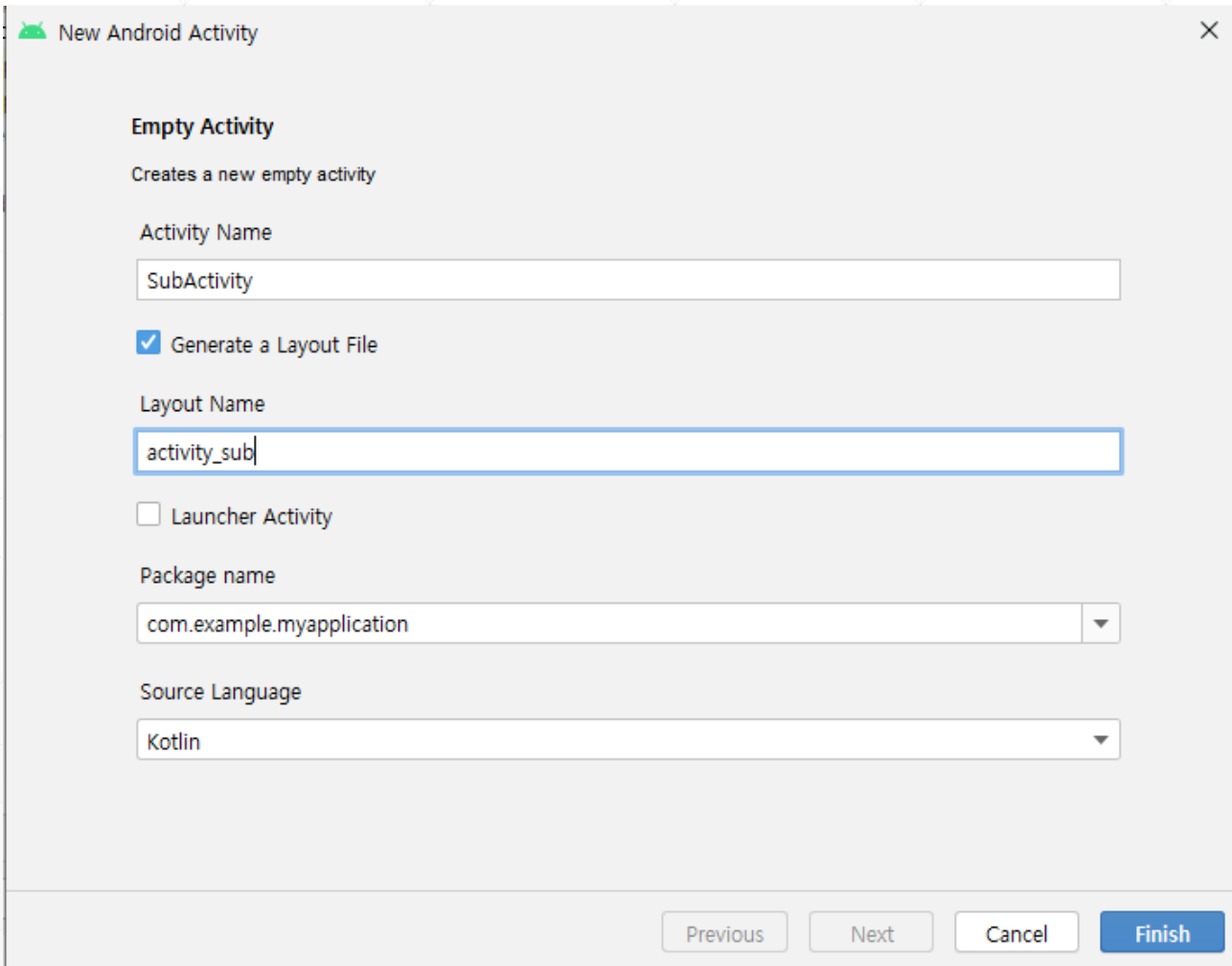
# Interacting with Other Activities (1/5)

- Let's create a new activity



# Interacting with Other Activities (2/5)

- Let's create a new activity!



The screenshot shows the 'New Android Activity' dialog box. It has a title bar with an Android icon and a close button. The main content area is titled 'Empty Activity' with a subtitle 'Creates a new empty activity'. It contains several input fields and checkboxes: 'Activity Name' with the text 'SubActivity', a checked checkbox for 'Generate a Layout File', 'Layout Name' with the text 'activity\_sub', an unchecked checkbox for 'Launcher Activity', 'Package name' with the text 'com.example.myapplication', and 'Source Language' with the text 'Kotlin'. At the bottom, there are four buttons: 'Previous', 'Next', 'Cancel', and 'Finish'.

New Android Activity

**Empty Activity**  
Creates a new empty activity

Activity Name  
SubActivity

☒ Generate a Layout File

Layout Name  
activity\_sub

☐ Launcher Activity

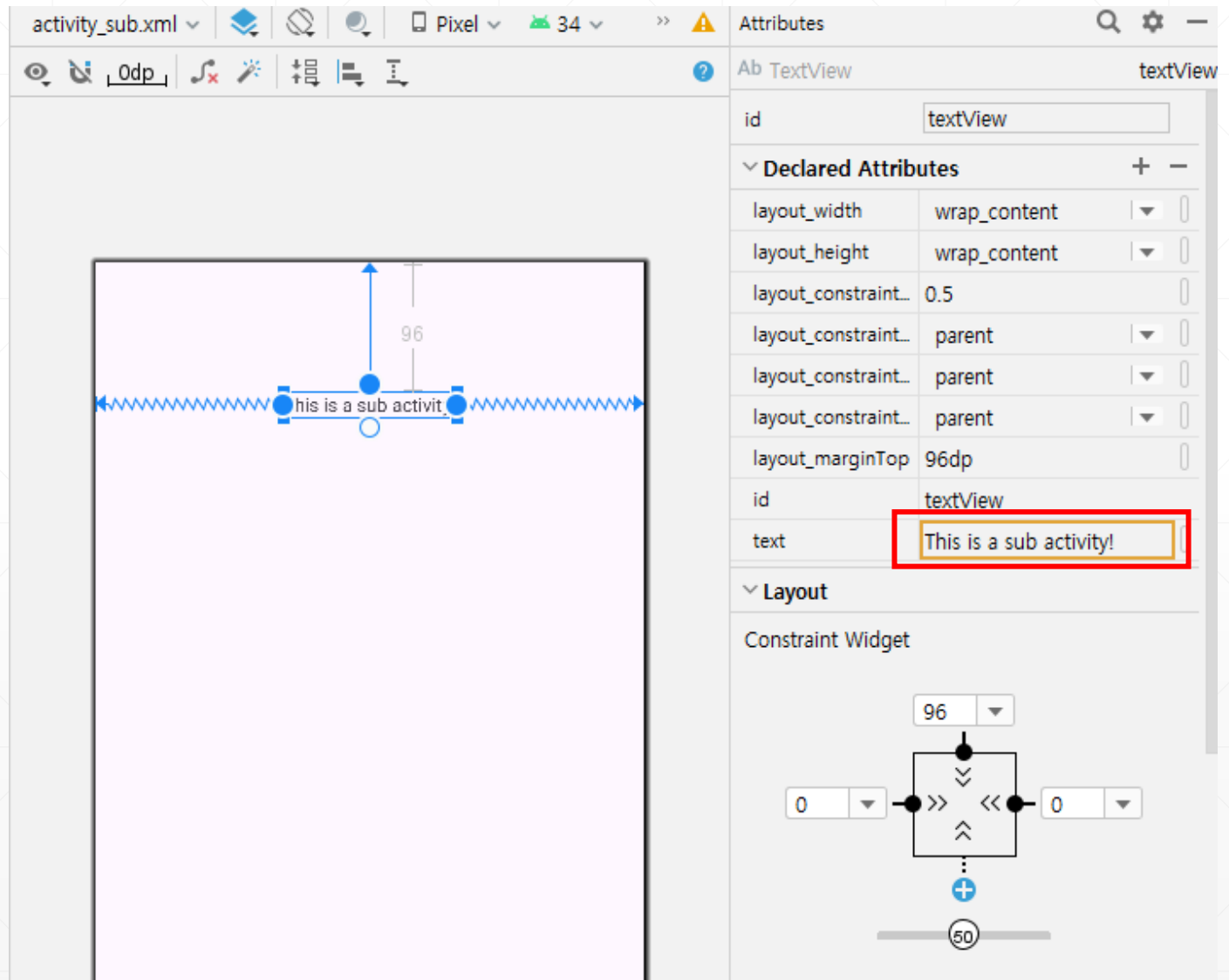
Package name  
com.example.myapplication

Source Language  
Kotlin

Previous Next Cancel Finish

# Interacting with Other Activities (3/5)

- Add TextView in the sub activity





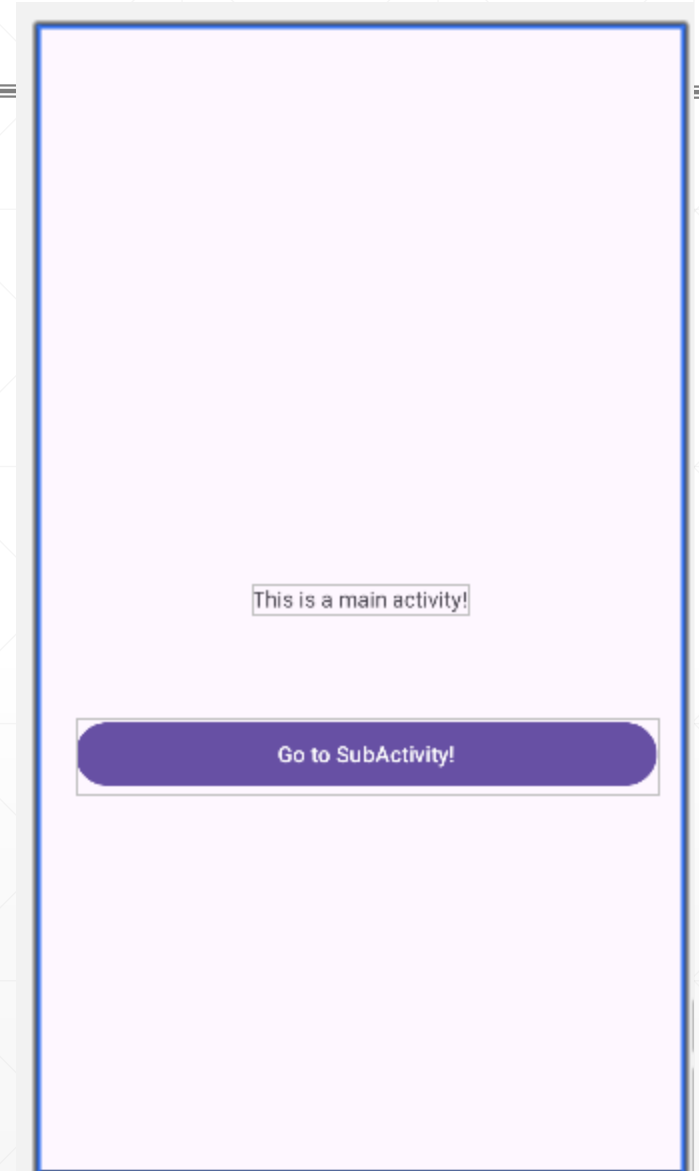
# Interacting with Other Activities (4/5)

## ■ Edit the UI widgets in the main activity

- Textview → “This is a main activity!”
- Button → “GO TO SUBACTIVITY”

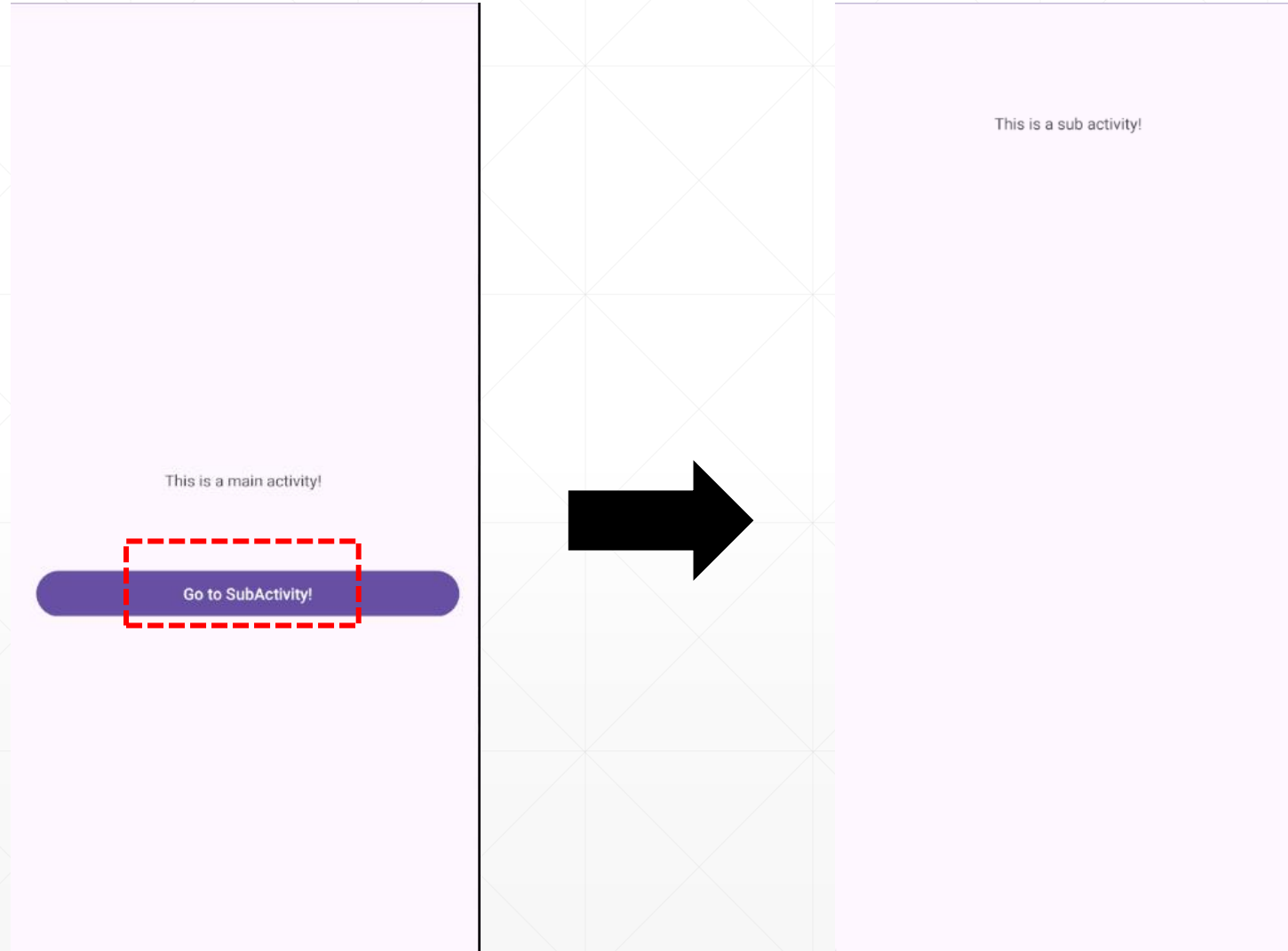
## ■ Code to start another activity

```
class MainActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        Log.d("ITM", "onCreate() called!")  
  
        val binding = ActivityMainBinding.inflate(layoutInflater)  
        setContentView(binding.root)  
  
        binding.btnSay.setOnClickListener{  
            val intent = Intent(this, SubActivity::class.java)  
            startActivity(intent)  
        }  
    }  
}
```



# Interacting with Other Activities (5/5)

■ Oh!



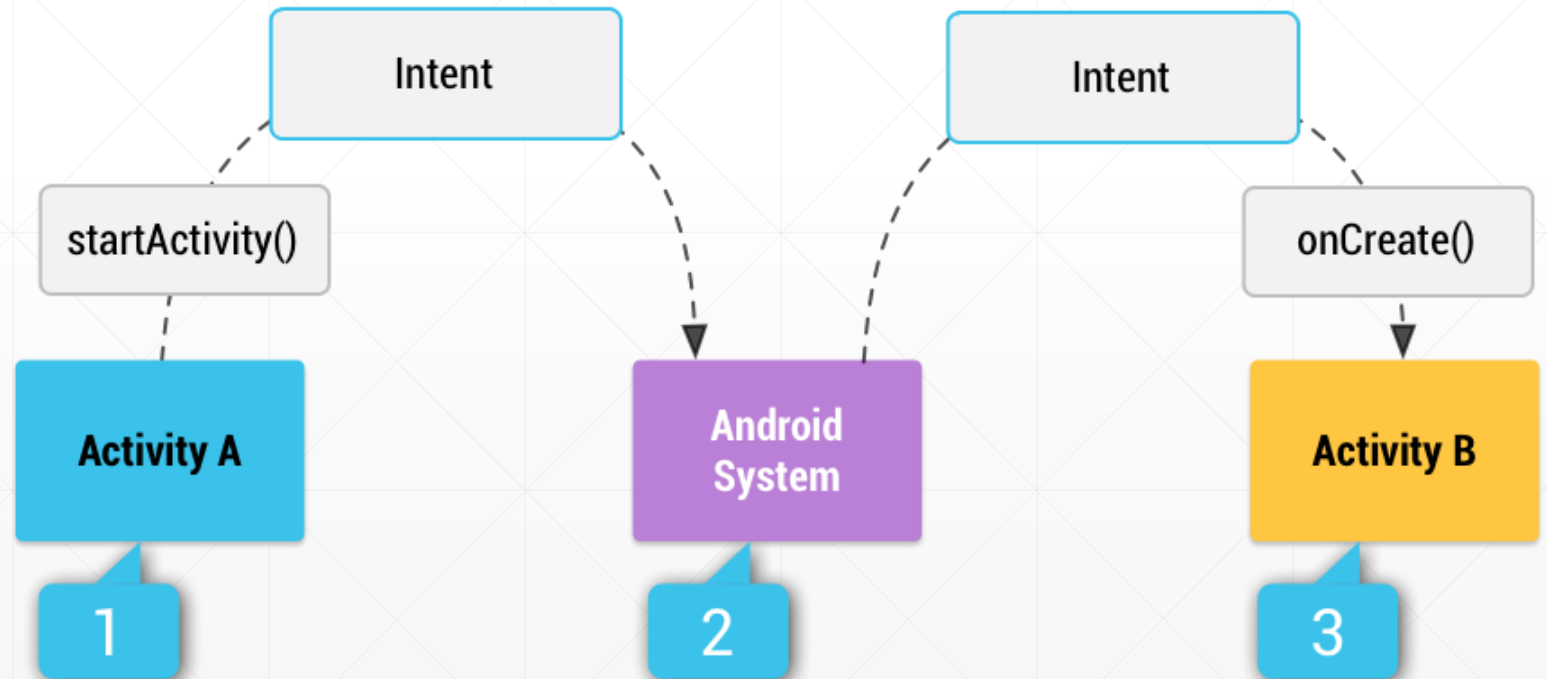
# Intent (1/4)

---

- Android app typically has several activities, each of which displays a user interface that allows the user to perform a specific task (such as view a map or take a photo)
- To take the user from one activity to another, your app must use an *Intent* to define your app's "intent" to do something!

# Intent (2/4)

- When you pass an Intent to the system with a method such as `startActivity()`, the system **uses the Intent to identify and start the appropriate app component**
- Using intents even allows your app to start an activity that is contained in a separate app!



# Intent (3/4)

---

## ■ Explicit intents

- **Specify which application will satisfy the intent**, by supplying either the target app's package name or a fully-qualified component 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!

## ■ Implicit intents

- **Declare a general action to perform**, which allows an activity from another app to handle it
- Example) if you want to show the user a location on a map, you can use an implicit intent to request that another capable app show a specified location on a map!

# Intent (4/4)

---

- Intent object carries information that
  - 1) the Android system uses to determine which component to start
  - 2) the recipient component uses in order to properly perform the action
- Building intents
  - Component name (for explicit), Action, Data, Extras, Category, Flag
- Common intents
  - <https://developer.android.com/guide/components/intents-common>
  - Several implicit intents that you can use to perform common actions!

# Intent: Building (1/3)

## ■ Component (optional)

- The name of the component to start
- If you need to start **a specific component (i.e., explicitly!)** in your app, you should specify the component name
- Without a component name, the intent is considered **implicit** and the system decides which component should receive the intent based on the other intent information

```
binding.btnSay.setOnClickListener{  
    val intent = Intent(this, SubActivity::class.java)  
    startActivity(intent)  
}
```

# Intent: Building (2/3)

## ■ Action

- A string that specifies the generic **action to perform** (such as view or pick)
- The action largely determines how the rest of the intent is structured, particularly the information that is contained in the *data* and *extras*!
- Where are actions?
  - Action constants defined in Intent class
  - Use your specific actions
- Example)
  - [ACTION\\_VIEW](#): Display the data to the user
  - [ACTION\\_EDIT](#): Provide explicit editable access to the given data



# Intent: Building (3/3)

## ■ Data

- The type of data supplied generally depends on the intent's action
  - URI (Uri object) that references the data to be acted on (use setData() method to set data)
  - MIME type of that data (text/plain, Image/jpeg, ...) (use setType() method to set type)
- Specifying the MIME type of your data helps the Android system find the best component to receive your intent

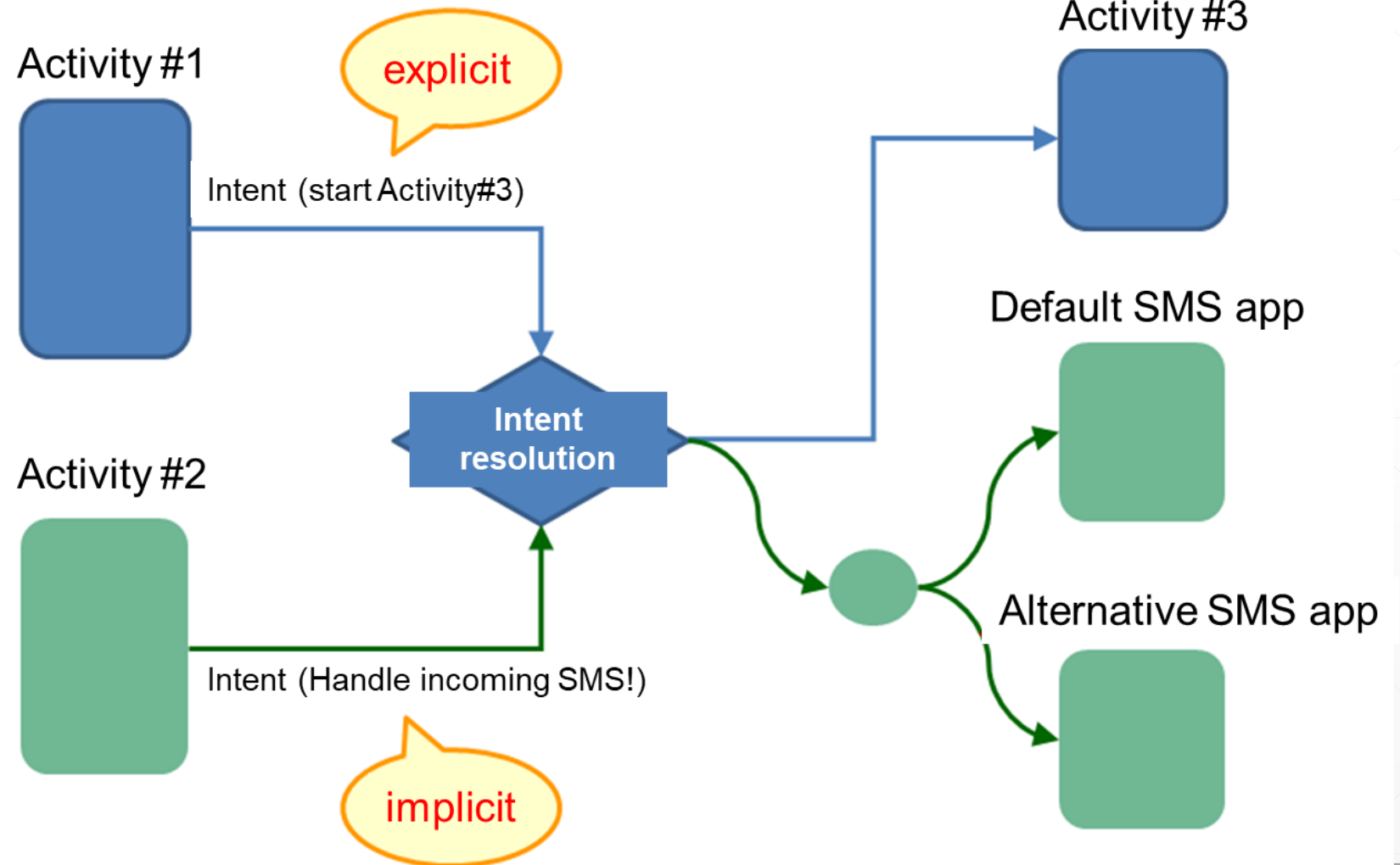
## ■ Extras

- Key-value pairs that carry additional information required to accomplish the requested action
- You can add extra data with various putExtra() methods, each accepting two parameters: the key name and the value
- You can also create a *Bundle* object with all the extra data, then insert the *Bundle* in the Intent with putExtras()

# Intent Resolution

- When the system receives an intent to start an activity, it searches for the best activity for the intent

➤ startActivity()



# Intent Example) (1/5)

## ■ Explicit intent

```
val intent = Intent(this, SubActivity::class.java).apply{  
    data = Uri.parse("http://seoultech.ac.kr")  
}  
binding.btnSay.setOnClickListener { startActivity(intent) }
```

MainActivity's onCreate()

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    val binding = ActivitySubBinding.inflate(layoutInflater)  
    setContentView(binding.root)  
  
    Log.d("ITM", intent.data.toString())  
}
```

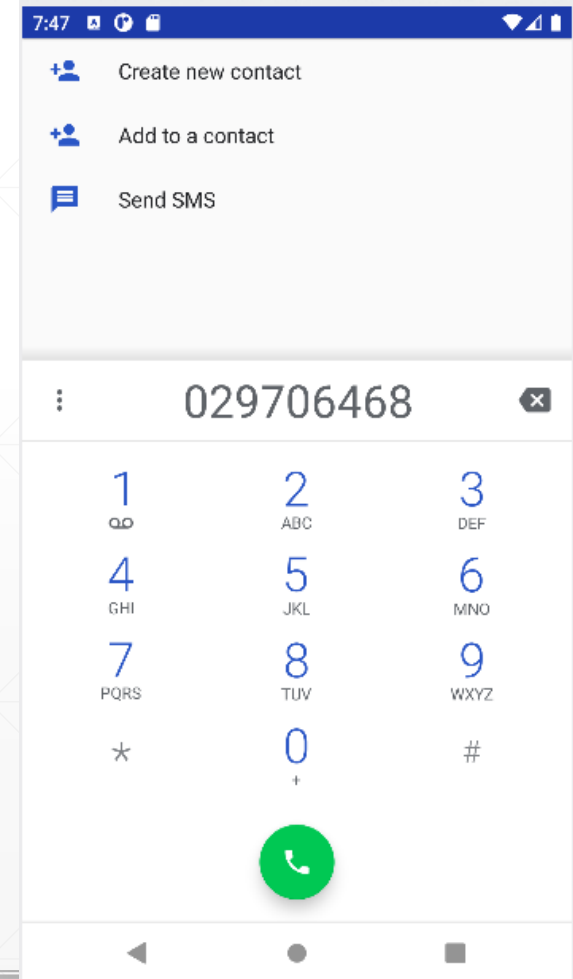
SubActivity's onCreate()

# Intent Example) (2/5)

## ■ Implicit intent ([ACTION\\_DIAL](#))

- Initiate a phone call
- Action
  - ACTION\_DIAL - Opens the dialer or phone app
- Data URI Scheme
  - tel:<phone-number>
  - voicemail:<phone-number>
- MIME Type
  - None

```
val intent = Intent(Intent.ACTION_DIAL).apply{  
    data = Uri.parse("tel:029706468")  
}  
binding.btnSay.setOnClickListener {  
    startActivity(intent) }
```



# Intent Example) (3/5)

## ■ Implicit intent ([ACTION\\_VIEW](#))

➤ Show a location on a map

➤ Data URI Scheme

- geo:latitude,longitude
  - Show the map at the given longitude and latitude ("geo:47.6,-122.3")
- geo:latitude,longitude?z=zoom
  - Show the map at the given longitude and latitude at a certain zoom level ("geo:47.6,-122.3?z=11")
- geo:0,0?q=lat,lng(label)
  - Show the map at the given longitude and latitude with a string label ("geo:0,0?q=34.99,-06.61(Treasure)")
- geo:0,0?q=my+street+address
  - Show the location for "my street address" (may be a specific address or location query ("geo:0,0?q=1600+Amphitheatre+Parkway%2C+CA"))

➤ MIME Type

- None

# Intent Example) (4/5)

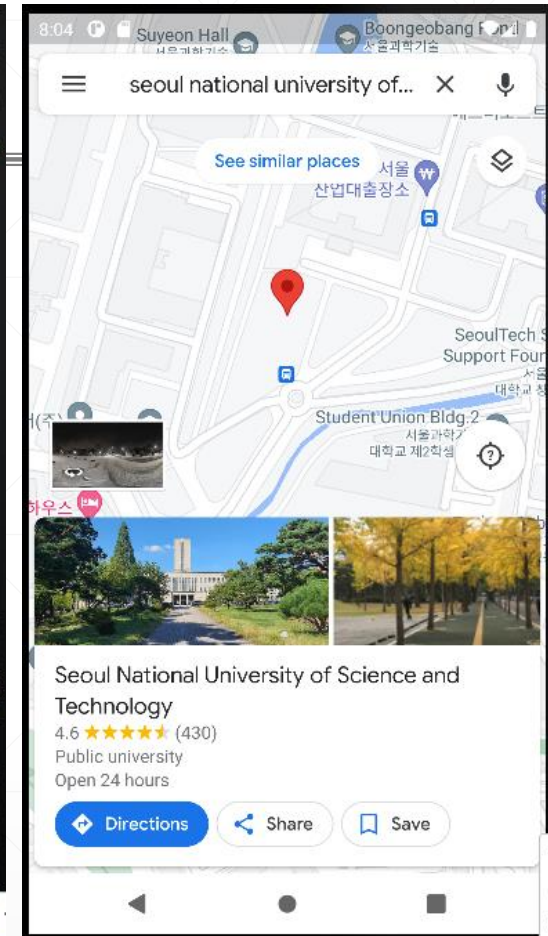
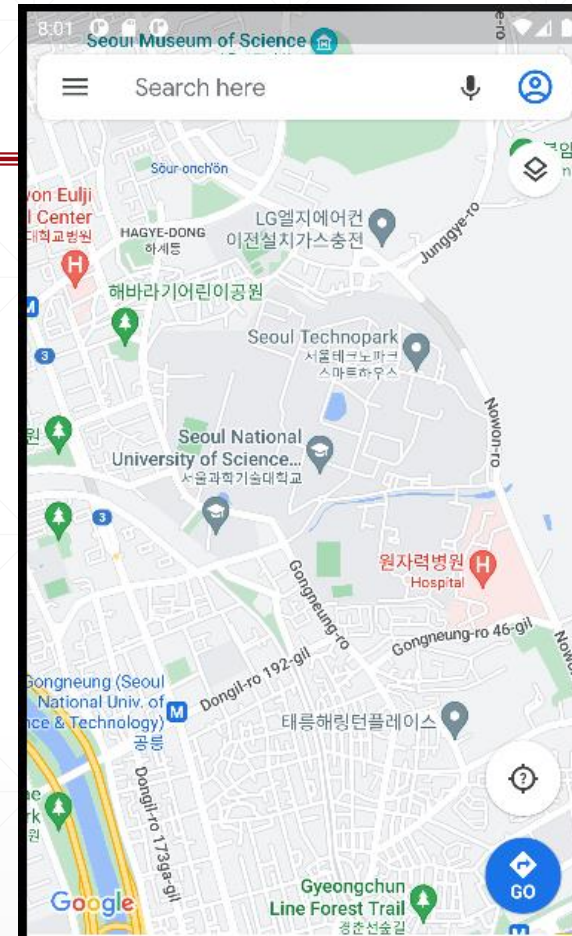
## ■ Implicit intent (ACTION\_VIEW)

- Show a location on a map

```
val intent = Intent(Intent.ACTION_VIEW).apply{  
    data = Uri.parse("geo:37.63177,127.077")  
}  
binding.btnSay.setOnClickListener { startActivity(intent) }
```

- Show a location on a map based on the address

```
val intent = Intent(Intent.ACTION_VIEW).apply{  
    data = Uri.parse("geo:0,0?q=seoul+national+university+of+science+and+technology")  
}  
binding.btnSay.setOnClickListener { startActivity(intent) }
```



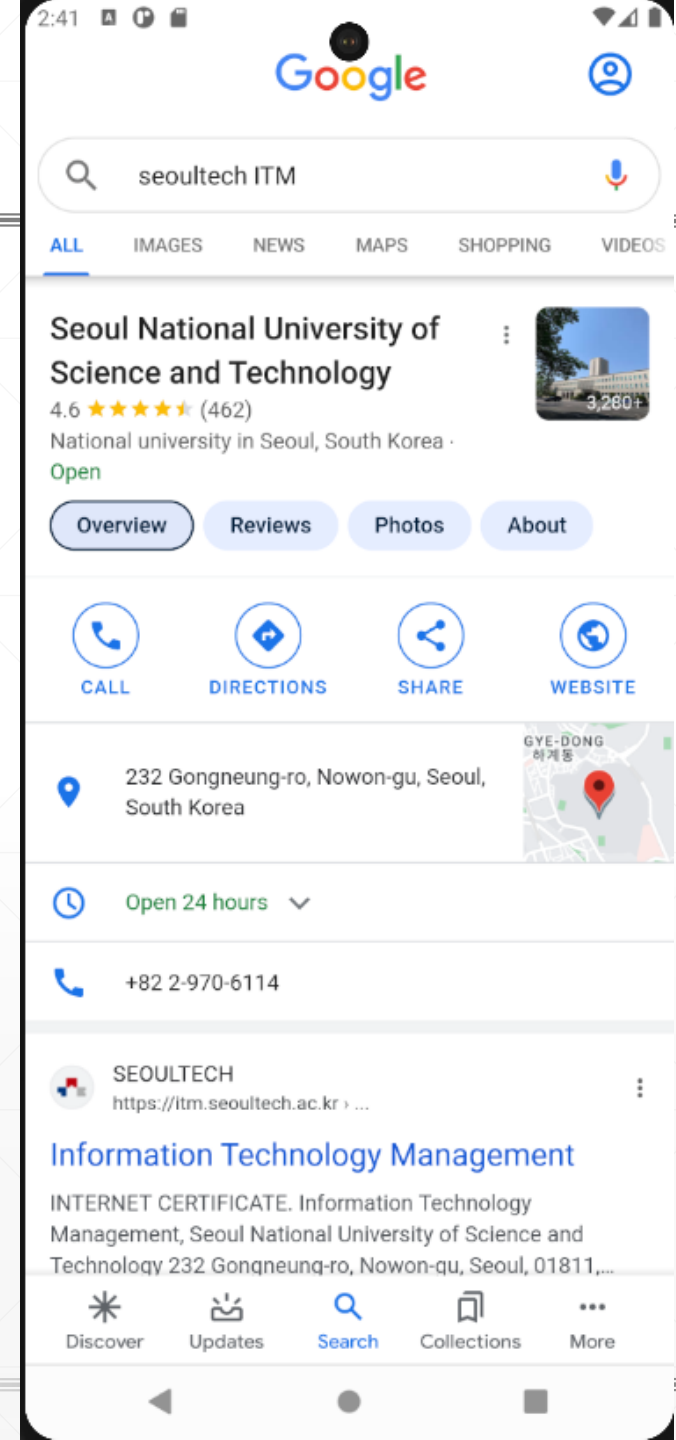


# Intent Example) (5/5)

## ■ Implicit intent ([ACTION\\_WEB\\_SEARCH](#))

- Initiate a web search
- Data URI Scheme
  - None
- MIME Type
  - None
- Extras
  - SearchManager.QUERY, search string

```
val intent = Intent(Intent.ACTION_WEB_SEARCH).apply{  
    putExtra(SearchManager.QUERY,"seoultech ITM")  
}  
binding.btnSay.setOnClickListener { startActivity(intent) }
```

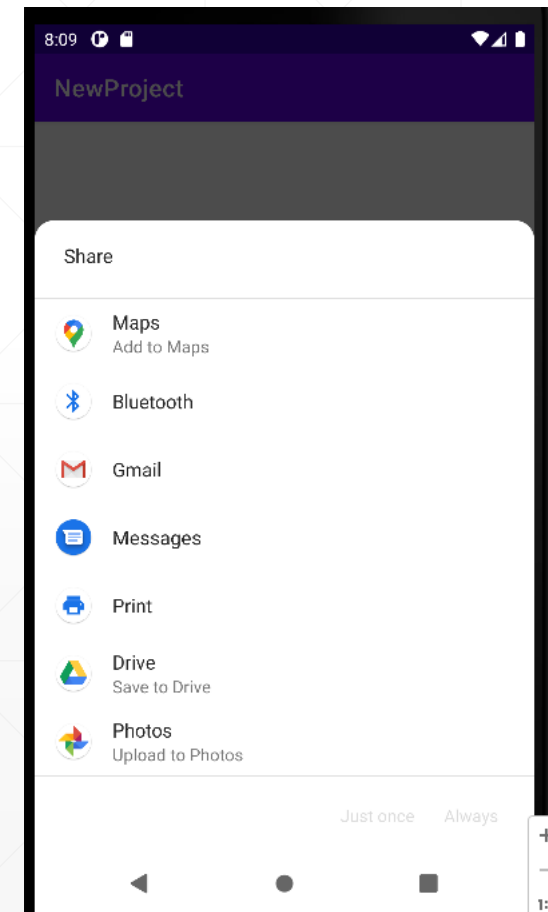


# Intent Resolution: Exceptions (1/3)

## ■ More than one matching application?

- User can select which app to use and make that app the default choice for the action

```
val intent = Intent(Intent.ACTION_SEND).apply {  
    type = "image/jpg"  
}  
binding.btnSay.setOnClickListener { startActivity(intent) }
```





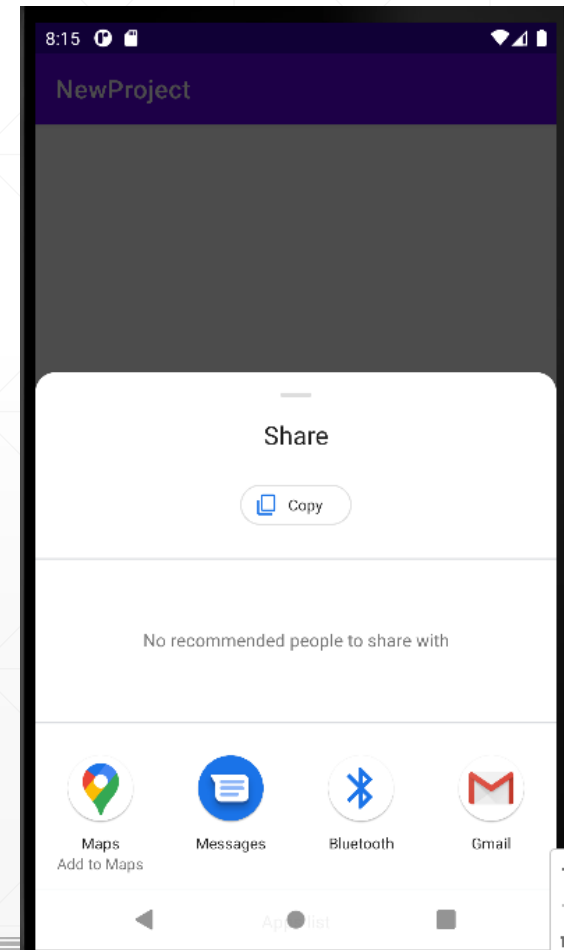
# Intent Resolution: Exceptions (2/3)

## ■ More than one matching application?

- If multiple apps can respond to the intent and the user might want to **use a different app each time**, then you should explicitly show an **app-chooser dialog!**
- To show the chooser, create an Intent using `createChooser()` and pass it to `startActivity()`
  - Setting a Title is working only when the target action is not `ACTION_SEND` or `ACTION_SEND_MULTIPLE`

```
val intent = Intent(Intent.ACTION_VIEW)
val chooser = Intent.createChooser(intent, "Show me the Picture!")
binding.btnSay.setOnClickListener { startActivity(chooser) }
```

```
val intent = Intent(Intent.ACTION_VIEW)
    .run { createChooser(this, "Show me the Picture!") }
binding.btnSay.setOnClickListener { startActivity(intent) }
```



# Intent Resolution: Exceptions (3/3)

## ■ No app can receive your intent?

- Your app should prepare for the situation **where no activity can handle your app's intent**
- Whenever you invoke an intent, be ready to catch an *ActivityNotFoundException*, which occurs if there's no other activity that can handle your app's intent
  - Alternatives: app-chooser, resolveActivity(), etc.

```
val intent = Intent("action_itm").apply {  
    type = "seoultech/ITM"  
}  
  
binding.btnSay.setOnClickListener {  
    try {  
        startActivity(intent)  
    } catch (e: ActivityNotFoundException) {  
        Log.d("ITM", "no apps found!")  
    }  
}
```

# Intent Resolution: Intent Filter (1/3)

## ■ How can be my app shown in the chooser?

- Add an `<intent-filter>` element in your **manifest file** for the corresponding `<activity>` element!
- Set **`android:exported`** flag to **True**

## ■ Define which intents your activity can handle

- Action: specify this in your intent filter with the `<action>` element
- Data: specify this in your intent filter with the `<data>` element
  - MIME type, just a URI prefix, just a URI scheme, or a combination of these
- Category: to receive implicit intents, **you must include the `CATEGORY_DEFAULT` category** in the intent filter

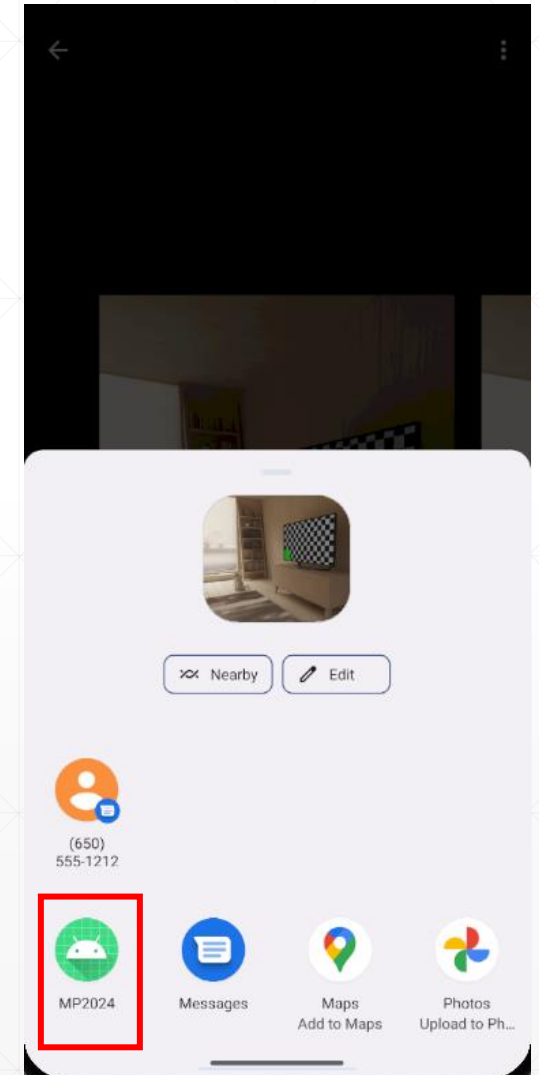
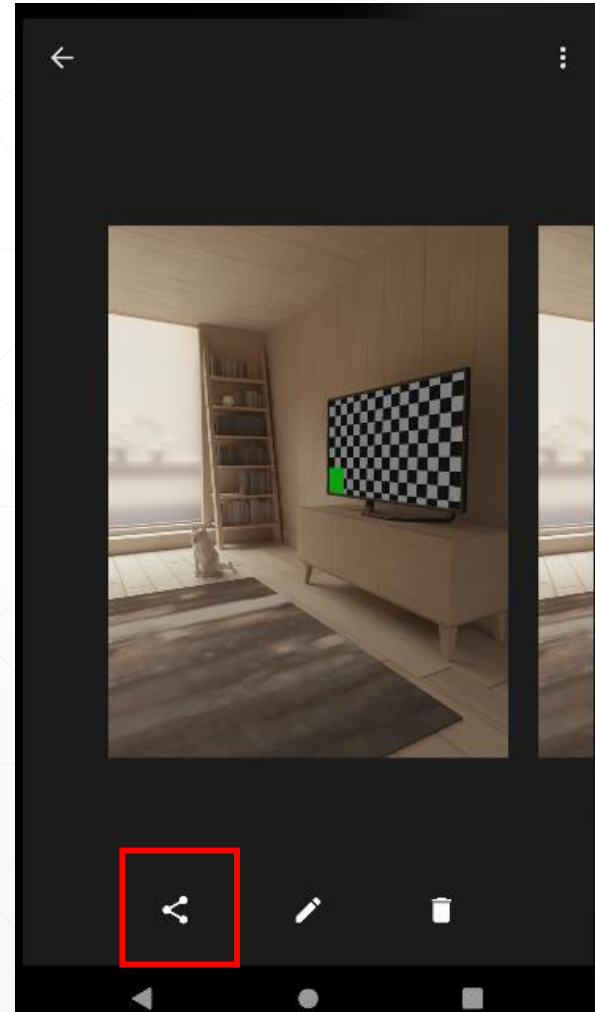
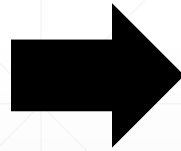
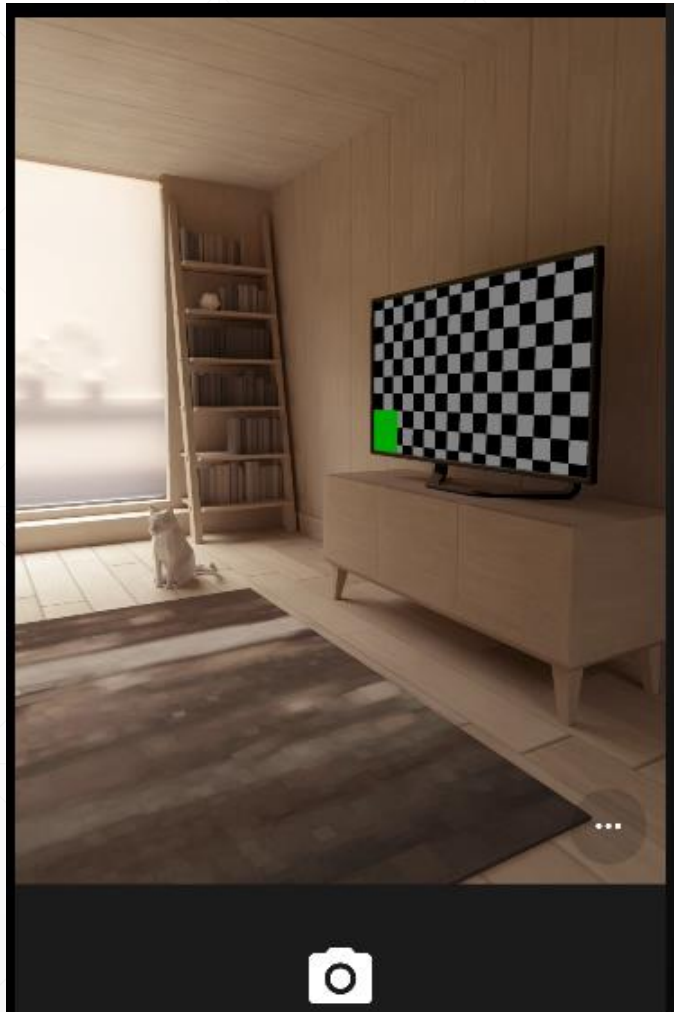
# Intent Resolution: Intent Filter (2/3)

## ■ Respond to ACTION\_SEND intent!

```
activity_main.xml x build.gradle.kts (:app) x MainActivity.kt x activity_sub.xml x SubActivity.kt
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
5 <application
8     android:icon="@mipmap/ic_launcher"
9     android:label="MP2024"
10    android:roundIcon="@mipmap/ic_launcher_round"
11    android:supportsRtl="true"
12    android:theme="@style/Theme.MP2024"
13    tools:targetApi="31">
14    <activity
15        android:name=".SubActivity"
16        android:exported="true">
17        <intent-filter>
18            <action android:name="android.intent.action.SEND"/>
19            <category android:name="android.intent.category.DEFAULT" />
20            <data android:mimeType="image/*"/>
21        </intent-filter>
22    </activity>
```

# Intent Resolution: Intent Filter (3/3)

■ Camera app → Take a photo → Share an image



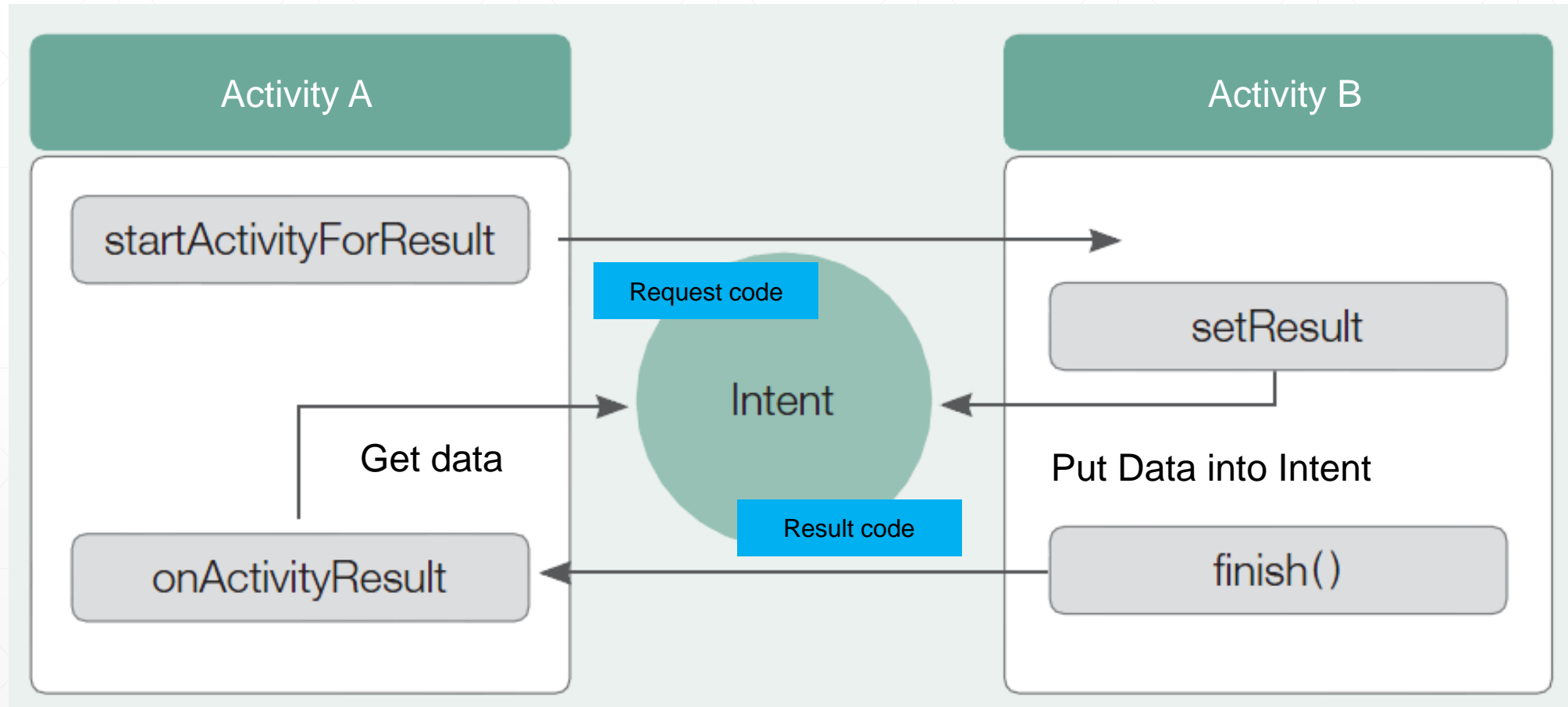
# Getting a Result from an Activity (1/7)

---

- Starting another activity does not need to be a one-way operation
- You can also start another activity and **receive a result back!**
- Example)
  - Your app can start a camera app and receive the captured photo as a result
  - You might start the Contacts app in order for the user to select a contact and receive the contact details as a result

# Getting a Result from an Activity (2/7)

## ■ Workflow (old way)



# Getting a Result from an Activity (3/7)

## ■ Caller methods (deprecated)

```
void startActivityForResult (Intent intent, int requestCode)
```

### Parameters

intent	Intent to send
requestCode	This code will be returned in onActivityResult() when the activity exits

```
void onActivityResult(int requestCode, int resultCode, Intent data)
```

### Parameters

requestCode	The integer request code originally supplied to startActivityForResult(), allowing you to identify who this result came from
resultCode	Result code (set by Callee activity by setResult() method)
data (optional)	An Intent, which can return result data to the caller (various data can be attached to Intent "extras")



# Getting a Result from an Activity (4/7)

## ■ Callee methods

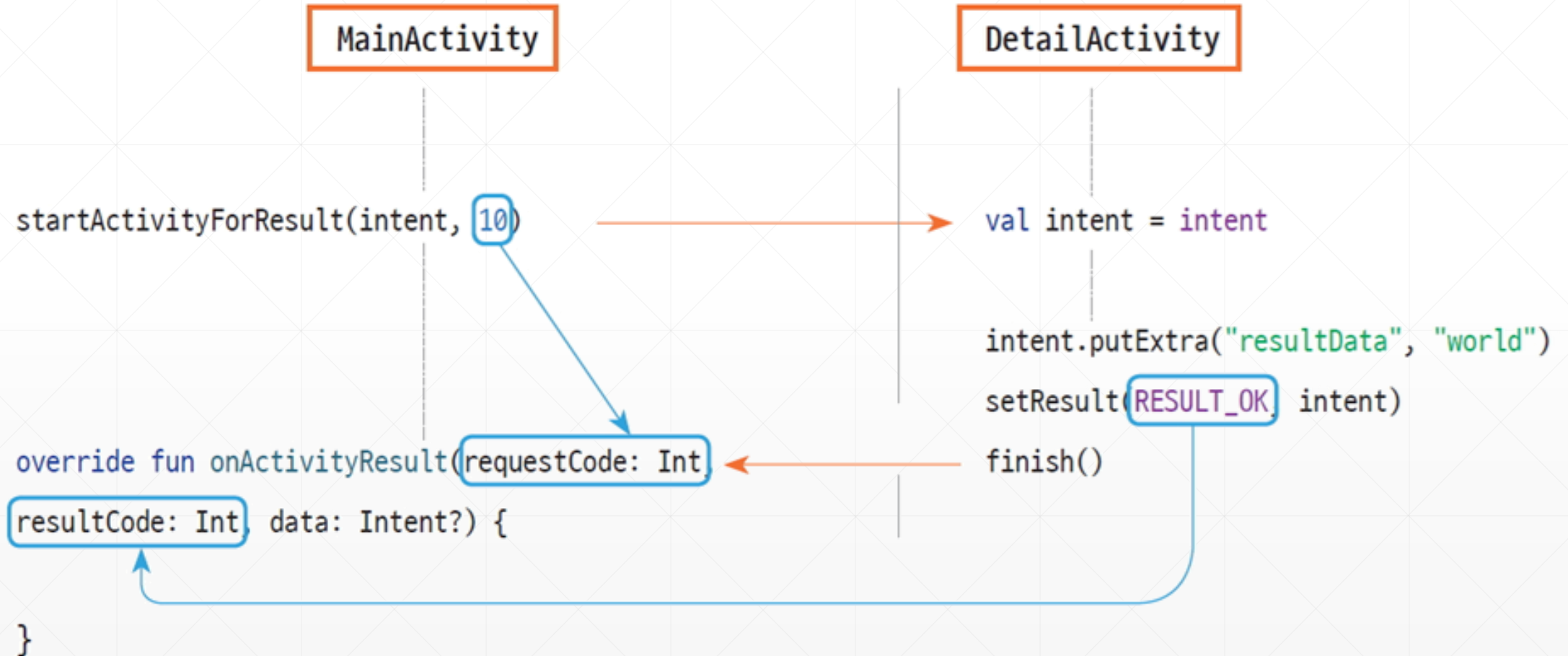
```
void setResult (int resultCode, Intent data)
```

### Parameters

resultCode	The result code to propagate back to the originating activity, often RESULT_CANCELED or RESULT_OK
data	The data to propagate back to the originating activity

# Getting a Result from an Activity (5/7)

## ■ Workflow (old way)



# Getting a Result from an Activity (6/7)

## ■ Example)

### - MainActivity

Start “SubActivity” activity

Get data named “grade”  
Display toast with the data

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    Log.d("ITM", "onCreated Called!")  
  
    val binding = ActivityMainBinding.inflate(layoutInflater)  
    setContentView(binding.root)  
  
    binding.btnSay.setOnClickListener {  
        val intent = Intent(this, SubActivity::class.java)  
        startActivityForResult(intent, 2024)  
    }  
}  
  
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {  
    super.onActivityResult(requestCode, resultCode, data)  
    Log.d("ITM", "requestCode: $requestCode resultCode: $resultCode")  
    Log.d("ITM", "${data?.getStringExtra("grade")}")  
    Toast.makeText(this, data?.getStringExtra("grade"), Toast.LENGTH_SHORT).show()  
}
```

# Getting a Result from an Activity (7/7)

## ■ Example)

### - SubActivity

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    val binding = ActivitySubBinding.inflate(layoutInflater)  
    setContentView(binding.root)
```

```
    Log.d("ITM", "SubActivity created!")  
    binding.btnReply.setOnClickListener {  
        intent.putExtra("grade", "${binding.txtReply.text}")  
        setResult(RESULT_OK, intent)  
        finish()  
    }  
}
```



# New Way for Getting a Result from an Activity

---

## ■ Activity Result API

- Provides components for registering for a result, launching the result, and handling the result

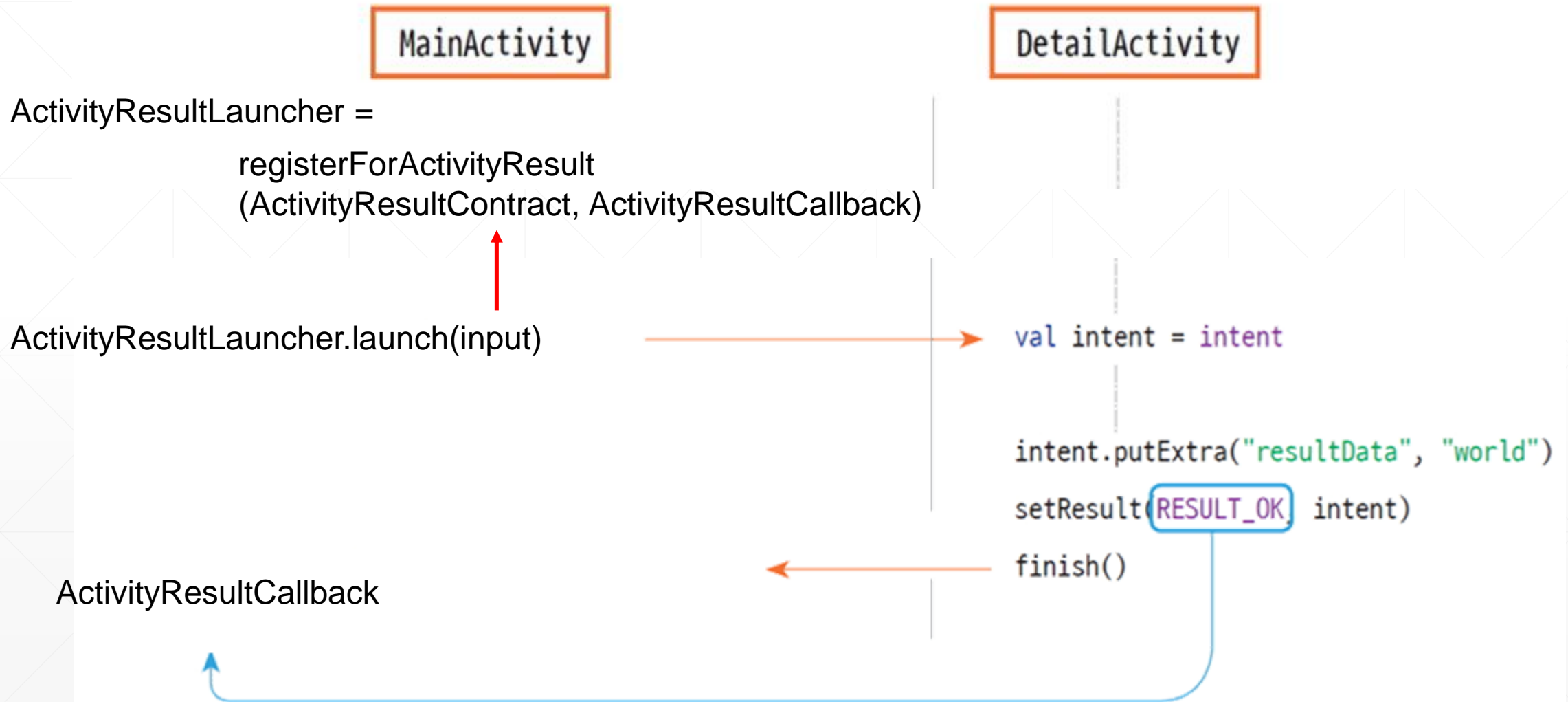
## ■ Step

- Registering a callback for an Activity Result
- Launching an activity for result
- Receiving an activity result

## ■ More details

- <https://developer.android.com/training/basics/intents/result>

# New Way for Getting a Result from an Activity



# New Way for Getting a Result from an Activity

## ■ registerForActivityResult()

- Takes an ActivityResultContract and an ActivityResultCallback
- Returns an ActivityResultLauncher which you will use to launch the other activity

## ■ ActivityResultContract

- Defines the **input type** needed to produce a result along with **the output type** of the result
- The APIs provide default contracts for basic intent actions like taking a picture, requesting permissions, and so on
- You can also create your own custom contracts

## ■ ActivityResultCallback

- Callback to be executed when activity result is available
- Takes an object of the output type defined in the ActivityResultContract

# New Way for Getting a Result from an Activity

---

## ■ Note

- If you have multiple activity result calls?
  - Call `registerForActivityResult()` multiple times to register multiple `ActivityResultLauncher` instances
- `registerForActivityResult()` is safe to call **before your fragment or activity is created**
  - Typically as a field initializer of an Activity or Fragment!



# New Way for Getting a Result from an Activity

## ■ ActivityResultLauncher

- registerForActivityResult() just registers your callback
- registerForActivityResult() **does not launch** the other activity and kick off the request

## ■ Calling **launch()** of ActivityResultLauncher starts the process of producing the result!

- If input exists, the launcher takes the input that matches the type of the ActivityResultContract
- When the user is done with the subsequent activity and returns, ActivityResultCallback is then executed

# New Way for Getting a Result from an Activity

## ■ StartActivityResult contract

- Takes raw Intent as an input
- Returns ActivityResult as an output

## ■ Example)

- Previous example with subactivity
- getContent() contract

## ■ More examples will be discussed later!

```
val requestLauncher = registerForActivityResult(ActivityResultContracts.StartActivityResult()){
    Log.d("ITM", "${it.resultCode}")
    Log.d("ITM", "${it.data?.getStringExtra("grade")}")
    Toast.makeText(this, it.data?.getStringExtra("grade"), Toast.LENGTH_SHORT).show()
}

val uriLauncher = registerForActivityResult(ActivityResultContracts.GetContent()){
    Log.d("ITM", it.toString())
}

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    val binding = ActivityMainBinding.inflate(layoutInflater)
    setContentView(binding.root)

    Log.d("ITM", "onCreated Called!")

    val intent = Intent(this, SubActivity::class.java)
    binding.btnSay.setOnClickListener { requestLauncher.launch(intent) } //
    uriLauncher.launch("image/*")
}
```