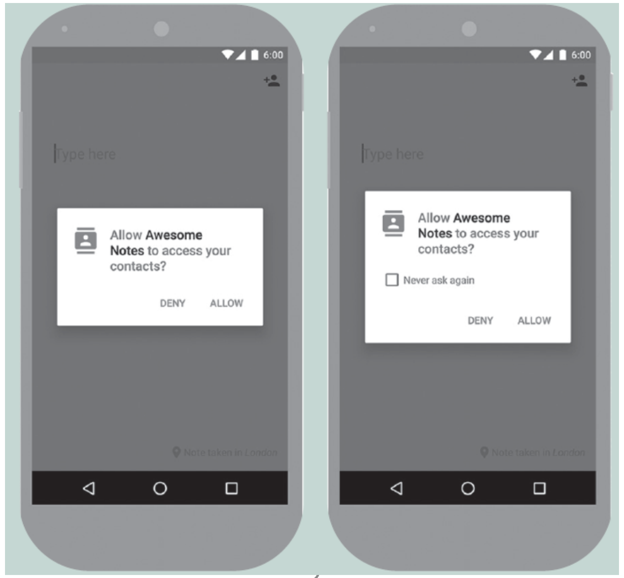
Application Development for Mobile Computer

<Week 10>

Youn Kyu Lee

Permission

- **How to Set Android App Permissions**
  - Write in the configuration file, AndroidManifest.xml
  - Write in the source code
    - If the permission you want to use risks exposing personal information, it is classified as a dangerous permission in the source code, and the permission must be delegated through a more complex method (e.g., a permission requested via a pop-up).
    - General permissions that do not directly expose personal information, such as internet access, should be written in the configuration file.


6:00
Type here
Allow Awesome Notes to access your contacts?
DENY
ALLOW
6:00
Type here
Allow Awesome Notes to access your contacts?
Never ask again
DENY
ALLOW

Permission Specification

- **Permission Specification**
  - Specifies whether certain data or functions can be used.
  - The AndroidManifest.xml file that defines permission specifications is located under the [app] → [manifests] directory.

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

Internet access permission

Wi-Fi access permission

- **Feature Specification**
  - Prevents downloading from the Play Store except on Android phones that have the corresponding feature
  - Even if not added separately to the AndroidManifest.xml file, the system automatically grants the permission when the corresponding feature is used
  - It is also possible to directly specify the feature to be used by using the <uses-feature/> tag

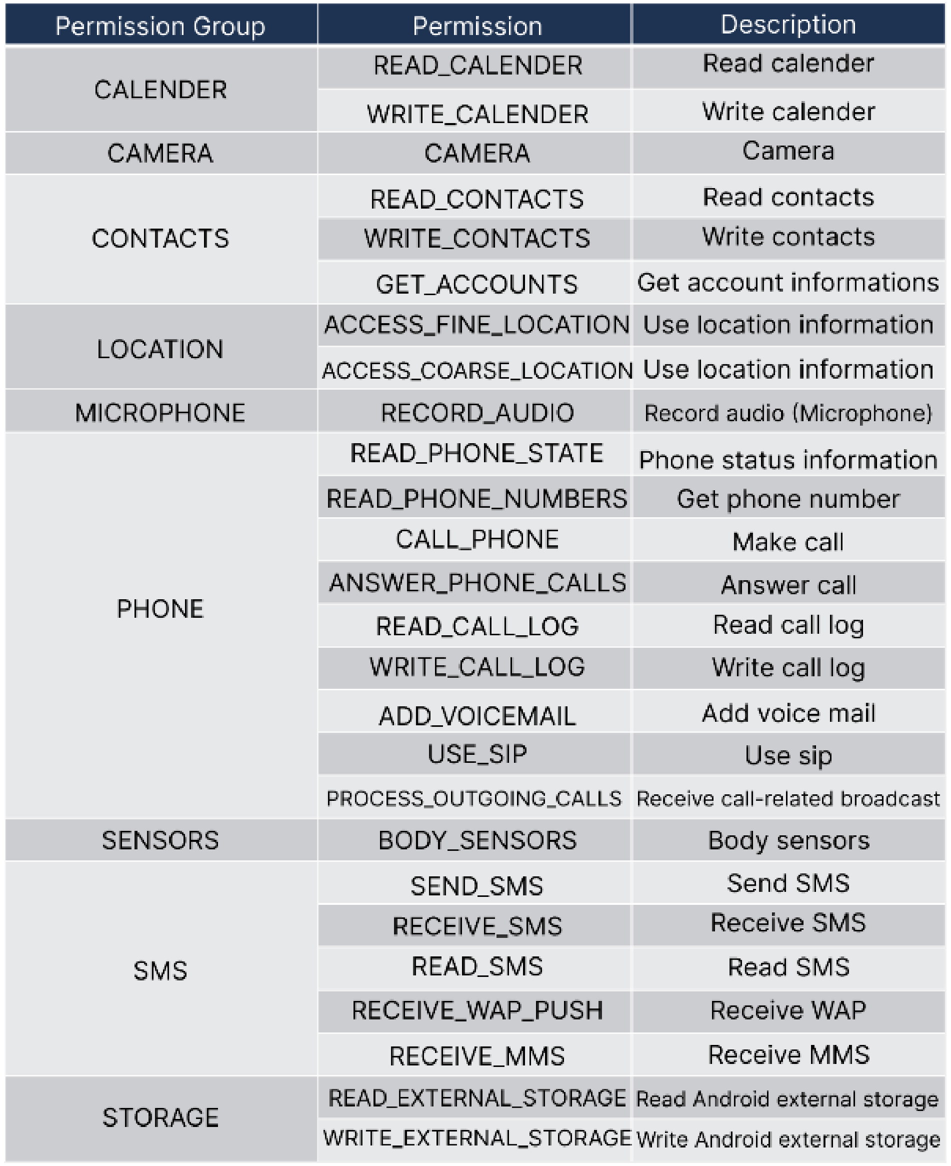
Level of Protection for Permission (1)

- **General Permission**

Permission	Description
ACCESS_NETWORK_STATE	Check network connection status
ACCESS_WIFI_STATE	Check Wi-Fi status
BLUETOOTH	Check Bluetooth status
INTERNET	Use network and internet
NFC	Use device-to-device short-range communication
SET_ALARM	Notification settings
VIBRATE	Vibration settings

Level of Protection for Permission (2)

- **Dangerous Permission**



Permission Group	Permission	Description
CALENDER	READ_CALENDER	Read calender
	WRITE_CALENDER	Write calender
CAMERA	CAMERA	Camera
CONTACTS	READ_CONTACTS	Read contacts
	WRITE_CONTACTS	Write contacts
	GET_ACCOUNTS	Get account informations
LOCATION	ACCESS_FINE_LOCATION	Use location information
	ACCESS_COARSE_LOCATION	Use location information
MICROPHONE	RECORD_AUDIO	Record audio (Microphone)
PHONE	READ_PHONE_STATE	Phone status information
	READ_PHONE_NUMBERS	Get phone number
	CALL_PHONE	Make call
	ANSWER_PHONE_CALLS	Answer call
	READ_CALL_LOG	Read call log
	WRITE_CALL_LOG	Write call log
	ADD_VOICEMAIL	Add voice mail
	USE_SIP	Use sip
	PROCESS_OUTGOING_CALLS	Receive call-related broadcast
SENSORS	BODY_SENSORS	Body sensors
SMS	SEND_SMS	Send SMS
	RECEIVE_SMS	Receive SMS
	READ_SMS	Read SMS
	RECEIVE_WAP_PUSH	Receive WAP
	RECEIVE_MMS	Receive MMS
STORAGE	READ_EXTERNAL_STORAGE	Read Android external storage
	WRITE_EXTERNAL_STORAGE	Write Android external storage

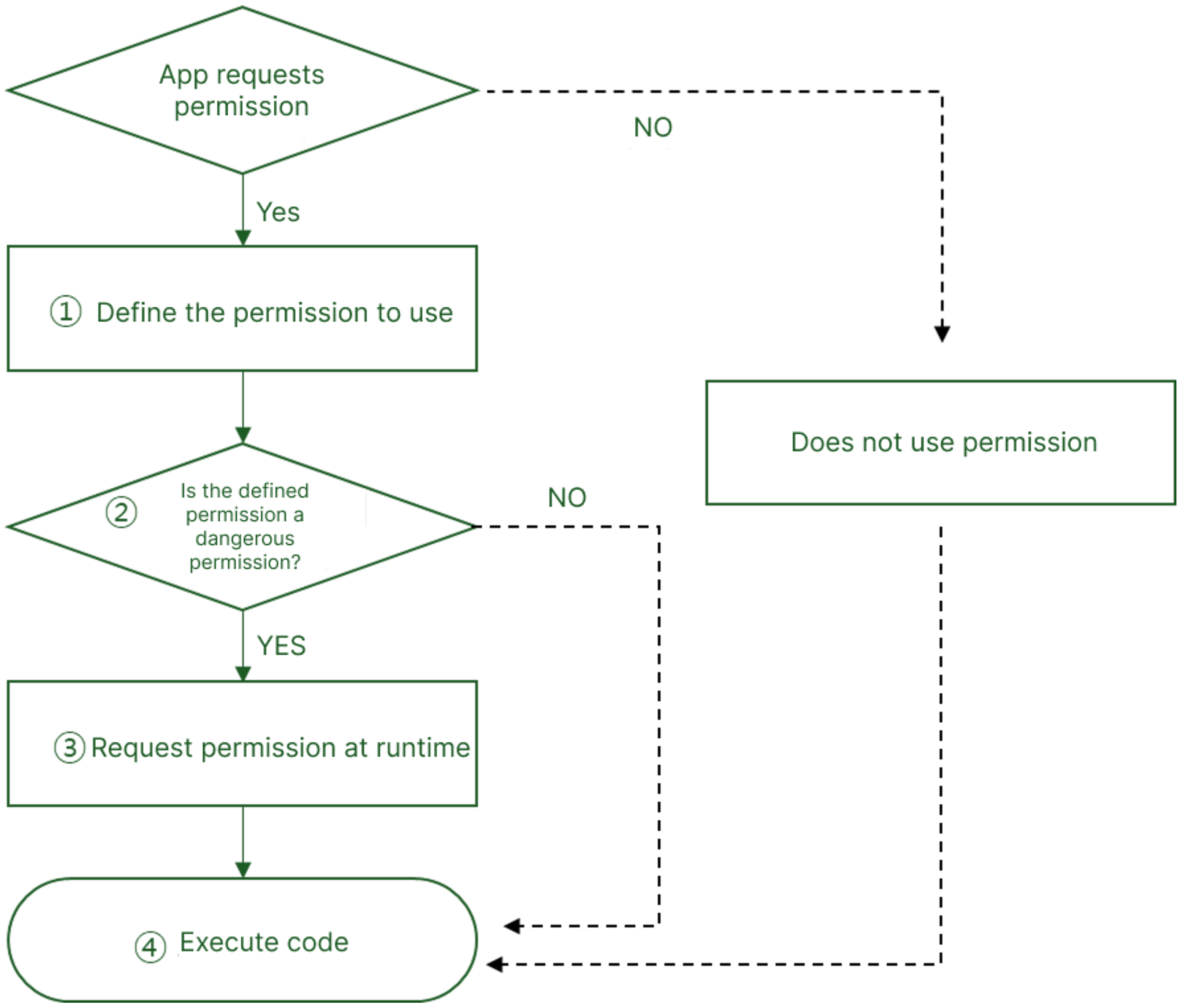
Level of Protection for Permission (3)

- **Signature Permission**
  - If the app requesting the permission is signed with the same certificate as the app that defines the permission, the system automatically grants that permission.
  - Some signature-level permissions cannot be used by third-party apps.

Permission Group

- **Permission Group**
  - Each permission is organized into groups.
  - For example, read and write permissions for files belong to the same group.
  - Permission requests are handled at the group level — if another permission in the same group has already been granted, the system automatically grants the new one.
  - Permissions within the same group are processed all at once.

Permission Request Processing Flowchart


App requests permission
NO
Yes
① Define the permission to use
Does not use permission
② Is the defined permission a dangerous permission?
NO
YES
③ Request permission at runtime
④ Execute code

Declaring in the configuration file AndroidManifest.xml

- **Handling Dangerous Permissions**
  - Declare in the AndroidManifest.xml Configuration File
    - Create a new project named Permission. In the build.gradle (Module:) file, check that viewBinding is enabled and add the dependency related to registerForActivityResult.

```
dependencies {
def dependency_version = "1.3.1"
implementation "androidx.activity:activity-ktx:$dependency_version"
implementation "androidx.fragment:fragment-ktx:$dependency_version"
// … omit
}
```

    - Open the AndroidManifest.xml file under the [app] → [manifests] directory and declare the required permissions.
    - Use the <uses-permission/> tag to add the camera permission. Below is an excerpt from AndroidManifest.xml showing the added camera permission:

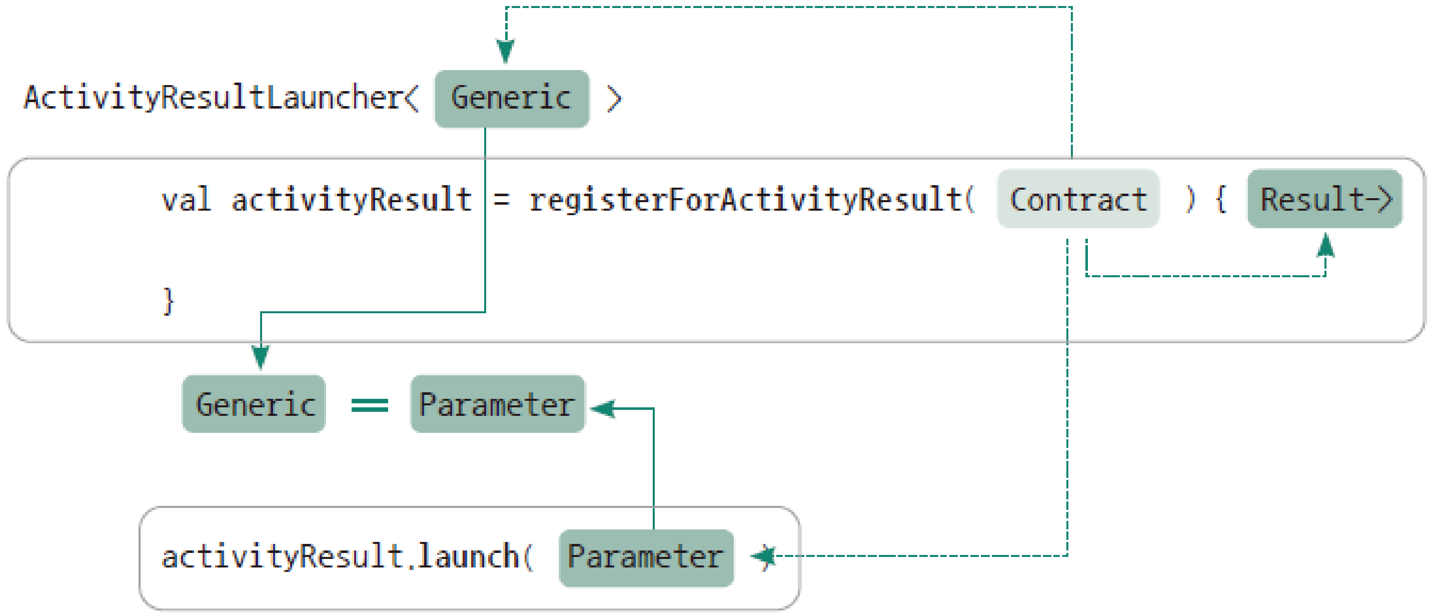
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="kr.co.hanbit.permission">

<uses-permission android:name="android.permission.CAMERA"/>    → Add this code
```

registerForActivityResult (1)

- **Handling Dangerous Permissions**
  - By using registerForActivityResult, you can not only handle results from your own activities but also access Andro id’s built-in features—such as the camera or gallery—depending on the type of contract you use.
  - When using a contract that handles permissions, the system displays a permission request pop-up to the user, an d your app can then process the approval or denial result.
  - The basic usage pattern is to create an ActivityResultLauncher using the registerForActivityResult() method.
  - When this launcher is executed, the code block written inside the method is automatically executed.

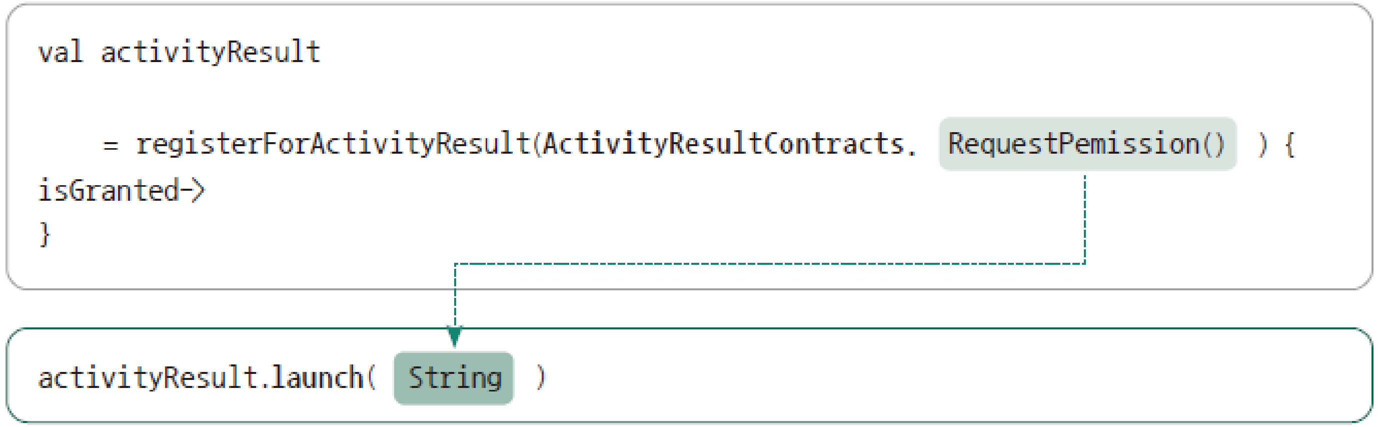
registerForActivityResult (2)

- **ActivityResultLauncher**
  - ActivityResultLauncher is created by calling registerForActivityResult() with a specified contract.
    - In the case of permission handling, you use either RequestPermission() or RequestMultiplePermissions() in the [Contract] section of the process.
    - A launcher is created by passing a predefined contract into registerForActivityResult().
    - When executed, the launcher triggers the corresponding activity or system feature (e.g., permission dialog).


ActivityResultLauncher< Generic >
val activityResult = registerForActivityResult(Contract) { Result->
}
Generic = Parameter
activityResult.launch(Parameter

registerForActivityResult (3)

- **ActivityResultContracts**
  - Depending on the type of Contract used, not only the generic type of the previously declared variable but also the parameter type passed to the launch() method is determined.
- **Contract for handling permissions**
  - When using RequestPermission() as a contract, you pass a single permission string as a parameter to the launch() method to request user approval.
  - When using RequestMultiplePermissions(), you pass an array of permissions. The result value, isGranted, can be renamed with an alias for better readability, but the default variable name it can also be used.

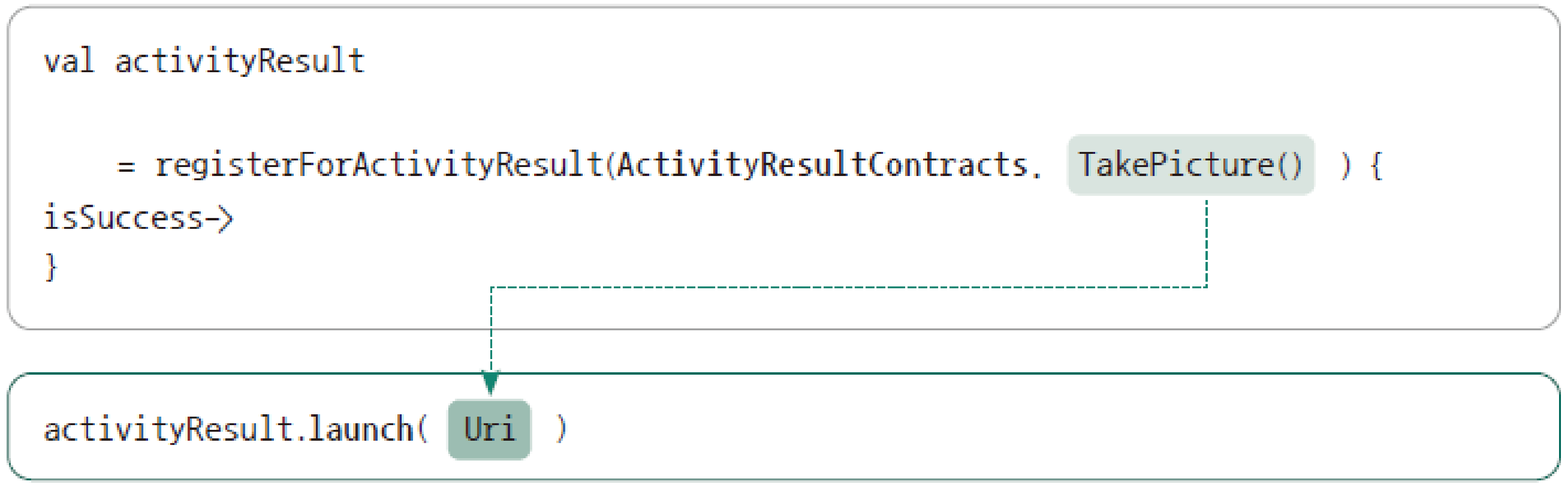


```
val activityResult

    = registerForActivityResult(ActivityResultContracts. RequestPemission() ) {
isGranted->
}
```

```
activityResult.launch( String )
```

registerForActivityResult (4)

- **Contract for Camera Call**

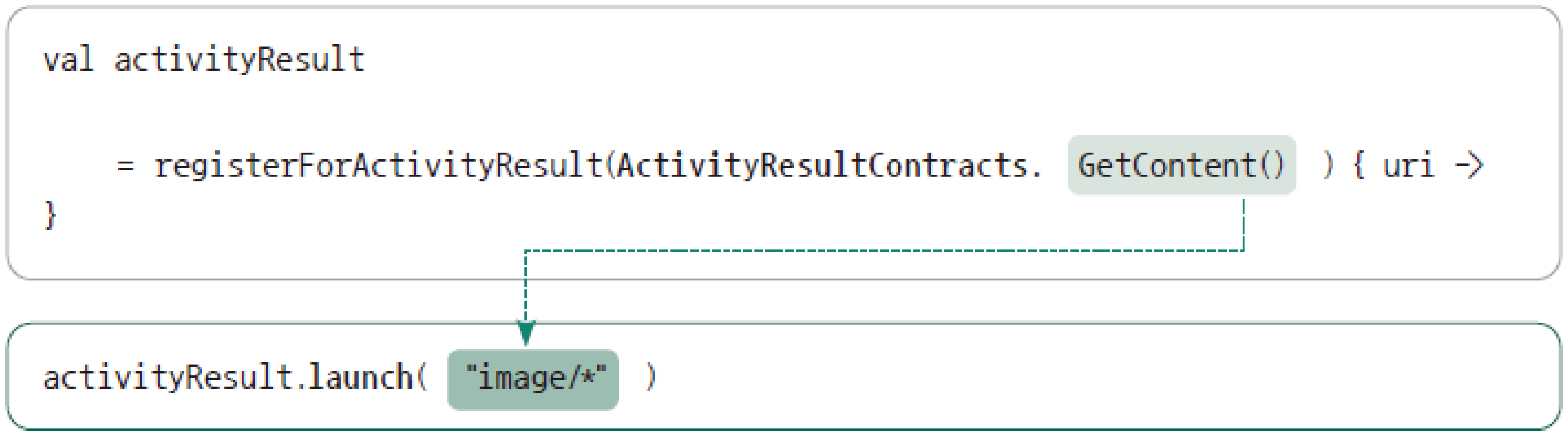


```
val activityResult

    = registerForActivityResult(ActivityResultContracts. TakePicture() ) {
isSuccess->
}
```

```
activityResult.launch( Uri )
```

registerForActivityResult (5)

- **Contract for Image Gallery Call**



```
val activityResult
    = registerForActivityResult(ActivityResultContracts. GetContent() ) { uri ->
}
```

```
activityResult.launch( "image/*" )
```

registerForActivityResult (6)

- **Types of AcitivityResultContracts**

CaptureVideo	Saves the video to the provided content Uri
CreateDocument	Displays a message to select a path to create a new document, returning the Uri of the created item
GetContent	Displays a message to select one piece of content
GetMultipleContents	Displays a message to select multiple pieces of content
OpenDocumen	Displays a prompt to open a document
OpenDocumentTree	Displays a message to select a directory and returns the selected one as a Uri
OpenMultipleDocuments	Requests to open multiple documents and receives the document contents as Uris
PickContact	Selects a contact from the contacts app
RequestMultiplePermissions	A contract for requesting multiple permissions
RequestPermission	A contract for requesting a permission
StartActivityForResult	A contract to receive the result value of an activity
StartIntentSenderForResult	A contract that calls StartIntentSender
TakePicture	A contract that saves the captured photo to a Uri
TakePicturePreview	A contract that returns a preview of the captured photo as a Bitmap

Handling Dangerous Permissions in Source Code (1)

- **Step 1: Create a permission request launcher**
  - First, write the basic app code and create a launcher for the permission request.
    1. Open the MainActivity.kt file and create the binding object. Save it in the binding property and pass binding.root to setContentView().
    2. Declare a property above the onCreate() method to handle permissions. In the following code, the contract RequestPermission() is used, and since it takes a single string as its parameter, the generic type is defined as <String>.

```
lateinit var activityResult:ActivityResultLauncher<String>
override fun onCreate( ) {
// ...  omit
}
```

    3. Inside the onCreate() method, create the launcher with registerForActivityResult() and store it in activityResult.

```
acti vityResult = registerForActivityResult(ActivityResultContracts.
RequestPermission()) { }
```

Handling Dangerous Permissions in Source Code (2)

4. Continue by adding the following code inside the registerForActivityResult() block. The result value will return true or false depending on whether the permission is granted. If the camera permission is granted (true), call the startProcess() method to launch the camera. If the permission is not granted (false), terminate the app.

```
registerForActivityResult(ActivityResultContracts
    .RequestPermission()) {
    if (it) {
        startProcess() // ①  If approved, proceed with the program
    } else {
        finish() // ②  If not approved, request the permission
    }
}
```

5. Create the startProcess() method below the onCreate() method. Inside startProcess(), write code to display a Toast message indicating that the camera is being launched.

```
fun startProcess() {
    Toast.makeText(this, "  Launch Camera  ", Toast.LENGTH_LONG).show())
}
```

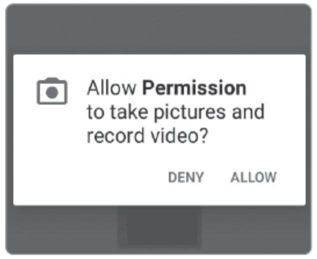
Handling Dangerous Permissions in Source Code (4)

- **Step 2: Request permission from the user**
  1. Below the launcher code written inside onCreate(), add code that executes the launcher when the camera button is clicked. Pass only one parameter (the camera permission) to the launch() function.

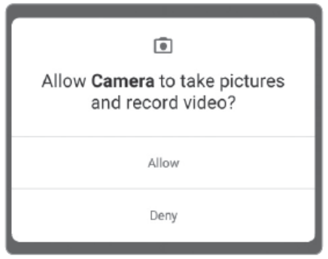
```
...

binding.btnCamera.setOnClickListener {
    activityResult.launch(Manifest.permission.CAMERA)
}
```

Import Android Package

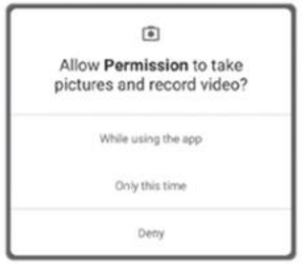
  2. Run the app and test it. When you click the camera button, a permission request popup appears. If approved, a Toast message saying "Launching camera" is displayed.



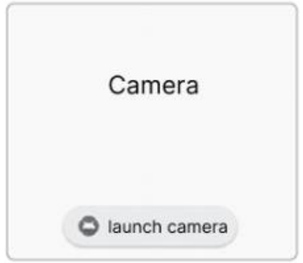
- v9.0 -



- v10.0 -



- v11.0, v12.0 -

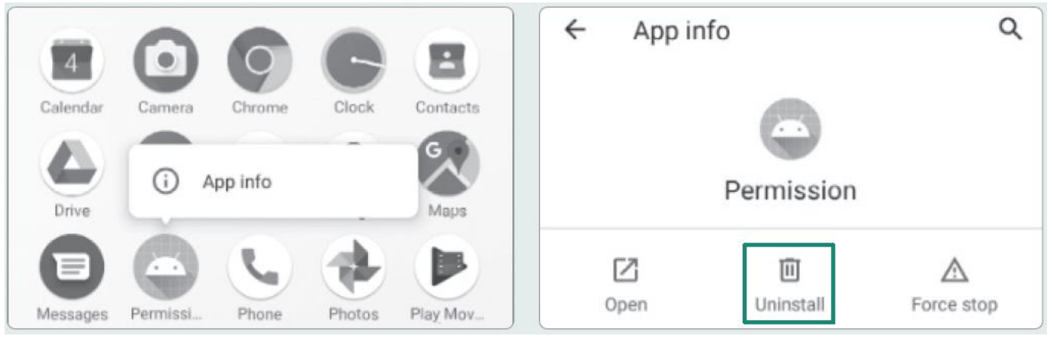


Handling Dangerous Permissions in Source Code (5)

- **Step 2: Request permission from the user**

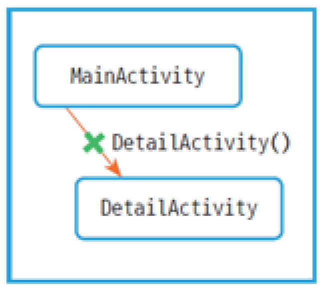
  3. After uninstalling and reinstalling the app, click Deny when the permission request appears — the app should terminate. If you don’t uninstall the app first, the previous permission state will remain stored, preventing proper testing.

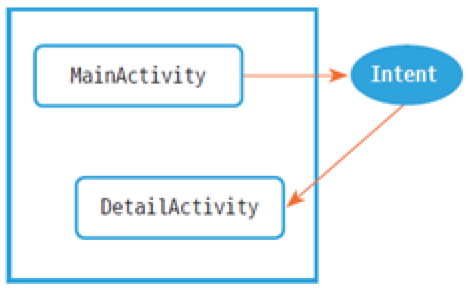
  - Deleting the App from the Emulator
    - Swipe up from the bottom of the emulator screen to open the app list.
    - Long-press the Camera app icon to open the App info menu.
    - Tap App info, then tap Uninstall to delete the app. (The position of the Uninstall button may vary depending on the emulator version.)

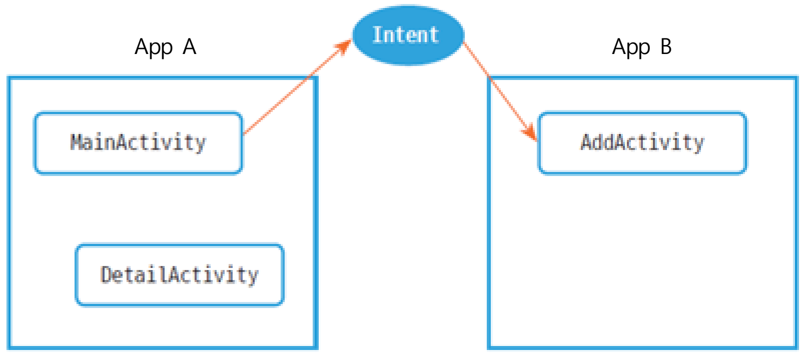


Intent (1)

- **What is an Intent?**
  - An Intent can be defined as a message sent to the system to request the execution of a component.
  - In Android, component classes cannot be executed directly by developers through code.
  - Instead, the system analyzes the information contained in the Intent and runs the corresponding component.
  - The same mechanism is used when linking with components from external apps.







Intent (2)

- **startActivity()**
  - The startActivity() function sends the Intent to the system.
  - The Intent constructor's parameter contains a class type reference that specifies which component to start.

• Registering MainActivity and DetailActivity

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

• Passing the Intent to the system

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

Intent (3)

- **Intent Extra Data**
  - When requesting the execution of a component with an Intent, you can send additional data using extra data.
  - Extra data refers to supplementary information stored within the Intent.
  - Use the putExtra() function to add data to an Intent.
  - Retrieve the data in the target component using functions such as getIntExtra().

- Put Extra Data

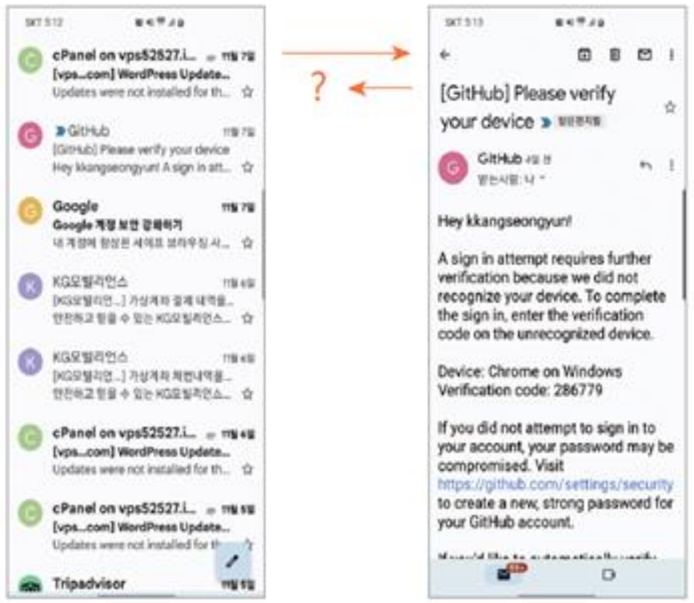
```kotlin
val intent: Intent = Intent(this, DetailActivity::class.java)
intent.putExtra("data1", "hello")
intent.putExtra("data2", 10)
startActivity(intent)
```

- Get Extra Data

```kotlin
val data1 = intent.getStringExtra("data1")
val data2 = intent.getIntExtra("data2", 0)
```

Intent (4)

- **Returning to the Previous Activity – ActivityResultLauncher**



cPanel on vps52527.i... 11월 7일
[vps...com] WordPress Update...
Updates were not installed for th...

GitHub 11월 7일
[GitHub] Please verify your device
Hey kkangseongyun! A sign in att...

Google 11월 7일
Google 계정 보안 강화하기
내 계정에 향상된 세이프 브라우징 사...

KG모빌리언스 11월 6일
[KG모빌리언...] 가상계좌 결제 내역을...
안전하고 믿을 수 있는 KG모빌리언스...

KG모빌리언스 11월 6일
[KG모빌리언...] 가상계좌 채번내역을...
안전하고 믿을 수 있는 KG모빌리언스...

cPanel on vps52527.i... 11월 6일
[vps...com] WordPress Update...
Updates were not installed for th...

cPanel on vps52527.i... 11월 5일
[vps...com] WordPress Update...
Updates were not installed for th...

Tripadvisor 11월 5일

[GitHub] Please verify your device

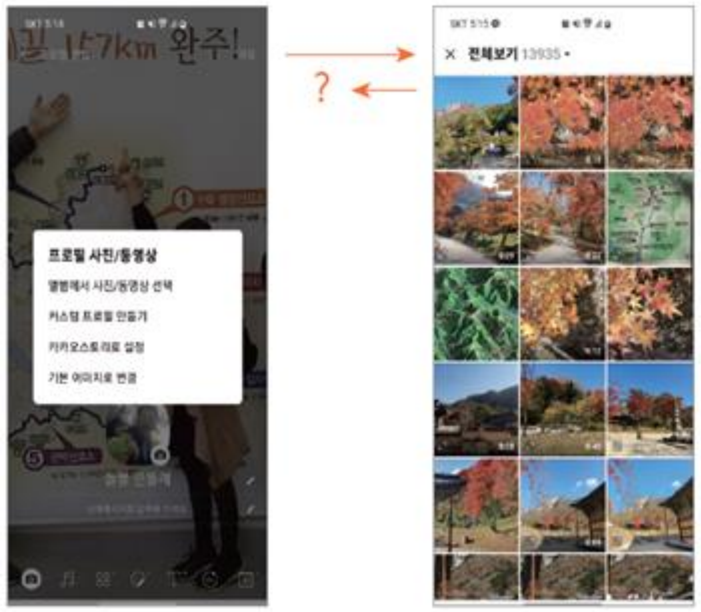
GitHub
받는사람: 나

Hey kkangseongyun!

A sign in attempt requires further verification because we did not recognize your device. To complete the sign in, enter the verification code on the unrecognized device.

Device: Chrome on Windows
Verification code: 286779

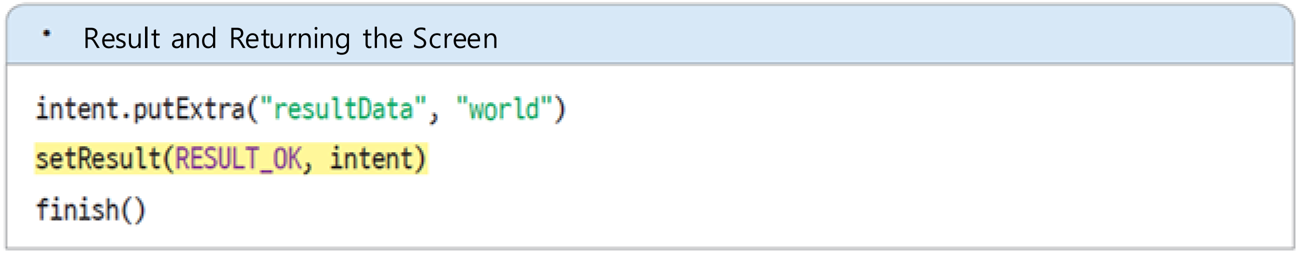
If you did not attempt to sign in to your account, your password may be compromised. Visit https://github.com/settings/security to create a new, strong password for your GitHub account.



Intent (5)

- **Starting an Activity with an Intent**
  - There are three main ways to start an Activity using an Intent:
    - public void startActivity(Intent intent)
    - public void startActivityForResult(Intent intent, int requestCode)
    - ActivityResultLauncher

- **Returning Results**
  - To automatically return to the previous screen, use the finish() function.

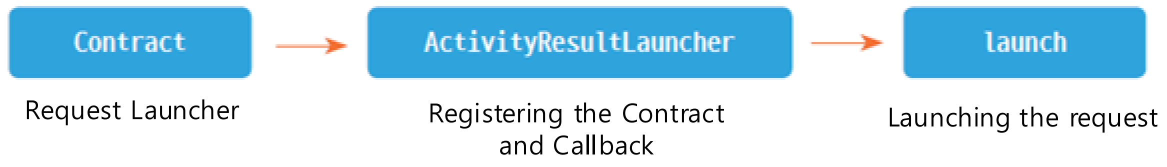


• Result and Returning the Screen

```
intent.putExtra("resultData", "world")
setResult(RESULT_OK, intent)
finish()
```

Intent (6)

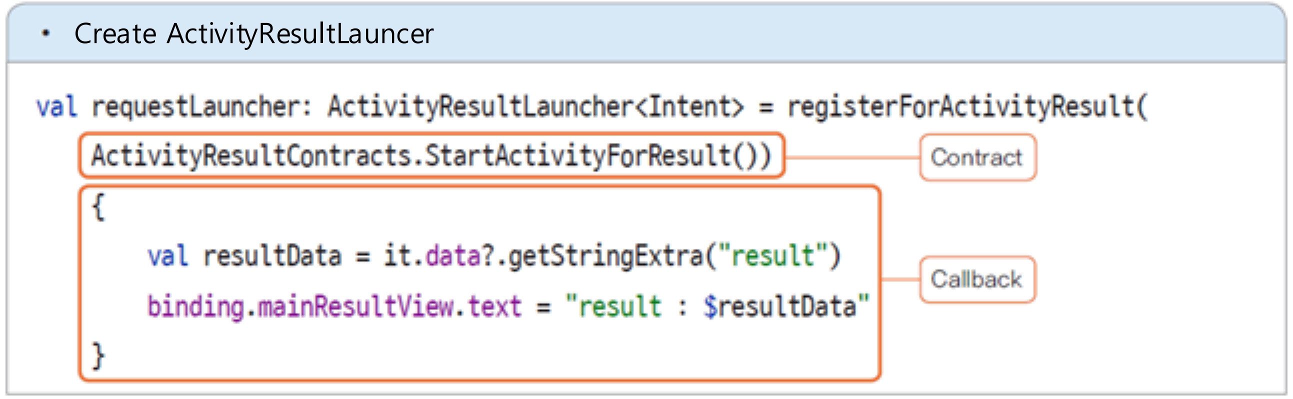
- **ActivityResultLauncher**



- **Contract**
  - A Contract defines the type of request handled by an ActivityResultLauncher.
  - It is a subclass that inherits from ActivityResultContract.
    - PickContact — Retrieves the URI of a selected contact
    - RequestPermission — Requests a single permission and checks if granted
    - RequestMultiplePermissions — Requests multiple permissions simultaneously
    - StartActivityForResult — Launches an activity and retrieves the result
    - TakePicturePreview — Captures an image and returns a bitmap preview
    - TakePicture — Captures, saves, and returns an image bitmap

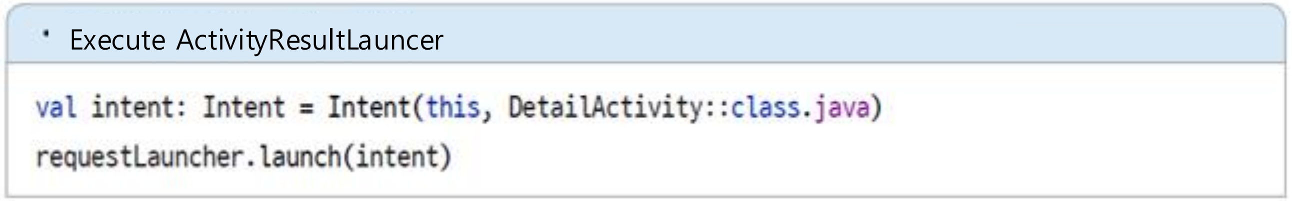
Intent (7)

- **ActivityResultLauncher**
  - ActivityResultLauncher is an object created using the registerForActivityResult() function.
  - This function takes two parameters: a Contract object (which defines the action to perform), and a Callback object (which handles the result).
- **Launch**
  - When the launch() function is called, the Contract registered in the ActivityResultLauncher is executed immediately.



- Create ActivityResultLauncer

```
val requestLauncher: ActivityResultLauncher<Intent> = registerForActivityResult(
    ActivityResultContracts.StartActivityForResult())
    {
        val resultData = it.data?.getStringExtra("result")
        binding.mainResultView.text = "result : $resultData"
    }
```

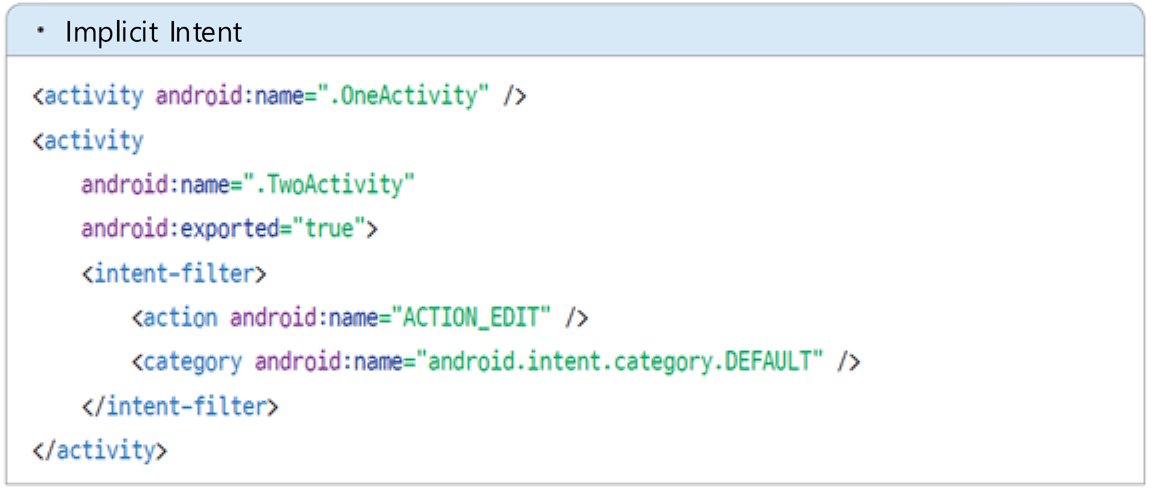


- Execute ActivityResultLauncer

```
val intent: Intent = Intent(this, DetailActivity::class.java)
requestLauncher.launch(intent)
```

Intent (8)

- **Intent Filter**
  - An Intent can be categorized into two types depending on how the target component is defined:
    - Explicit Intent: Uses a class type reference to specify the target component.
    - Implicit Intent: Uses information defined in an Intent Filter.
  - Explicit intents directly reference a class type, whereas implicit intents rely on the intent filters declared in the AndroidManifest.xml file.

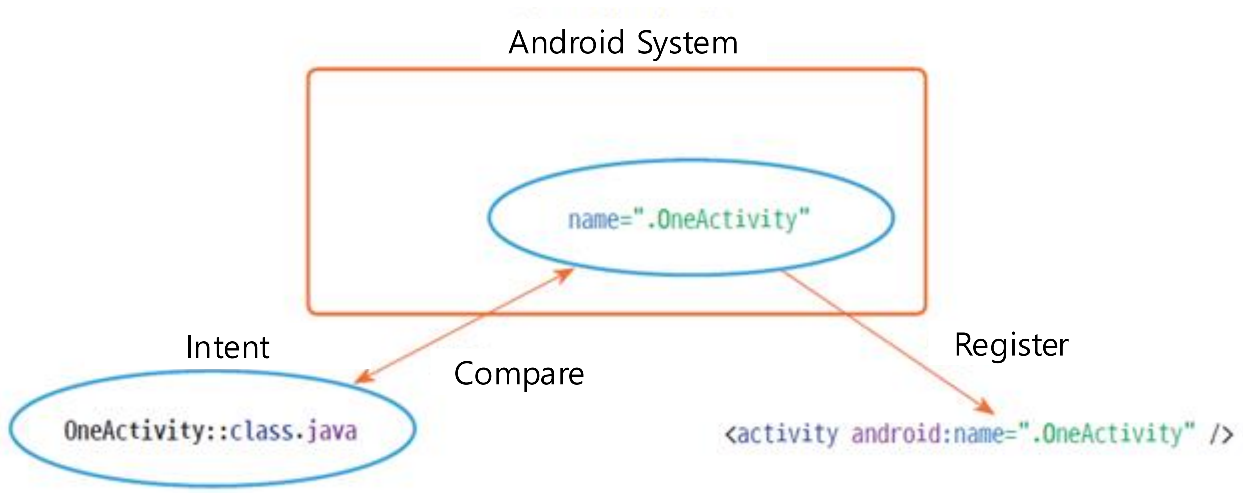


- Implicit Intent

```
<activity android:name=".OneActivity" />
<activity
    android:name=".TwoActivity"
    android:exported="true">
    <intent-filter>
        <action android:name="ACTION_EDIT" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

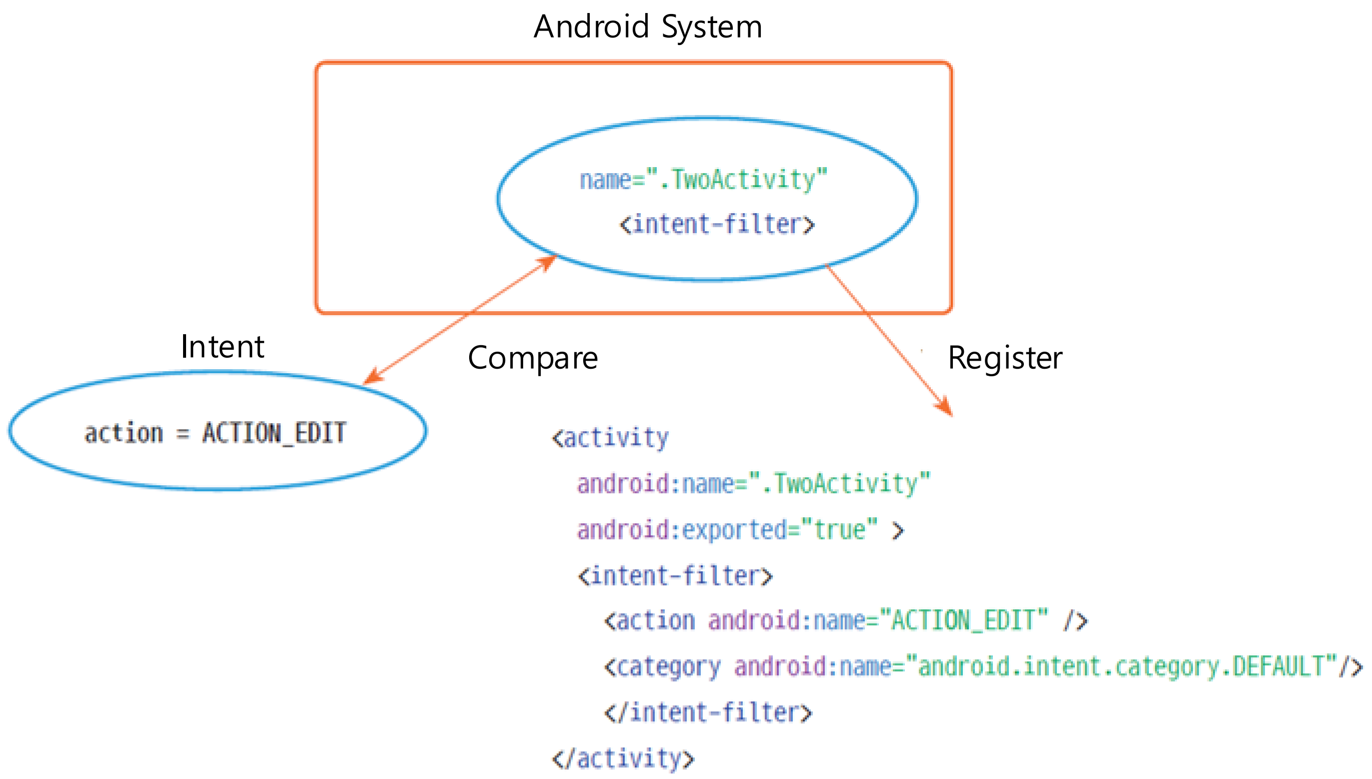
Intent (9)

- **Explicit Intent**



Intent (10)

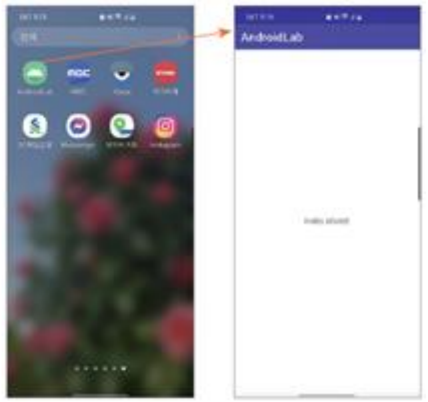
- **Implicit Intent**



```
<activity
  android:name=".TwoActivity"
  android:exported="true" >
  <intent-filter>
    <action android:name="ACTION_EDIT" />
    <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
</activity>
```

Intent (11)

- **Intent Filter**
  - Under an Intent Filter, information can be defined using the <action>, <category>, and <data> tags.
  - Which elements to include is up to the developer.
    - <action>: A string that represents the function or action of the component.
    - <category>: A string that specifies the category or context of the component.
    - <data>: Defines the data information required by the component.

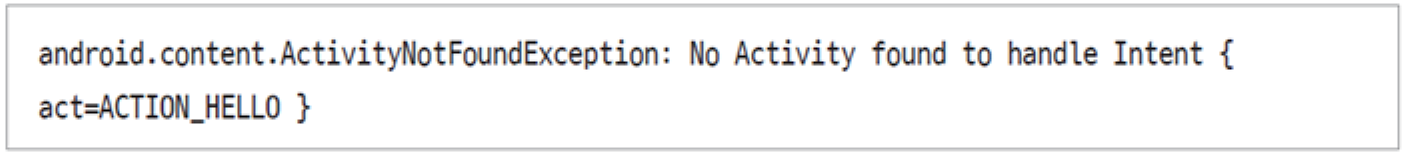


- Automatically generated Main Activity

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

Intent (12)

- **How Activity Intents Work**
  - How the system handles an intent depends on whether there are zero, one, or multiple target activities:
    - No activity available: An error occurs at the point where the intent is started.
    - One matching activity: The activity runs without any issue.
    - Multiple matching activities: The system prompts the user to choose one activity to launch.
  - If there is no available activity, an error similar to the following occurs.



```
android.content.ActivityNotFoundException: No Activity found to handle Intent {
act=ACTION_HELLO }
```

Intent (13)

- Consider the case where no activity exists to handle the intent.

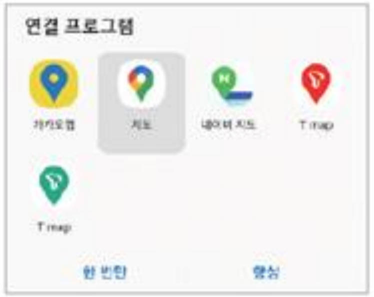
- Handle the exception when the corresponding Activity is not found

```
val intent = Intent("ACTION_HELLO")
try {
    startActivity(intent)
} catch (e: Exception) {
    Toast.makeText(this, "no app...", Toast.LENGTH_SHORT).show()
}
```

- If multiple activities can handle the intent, the system allows the user to choose one to execute.

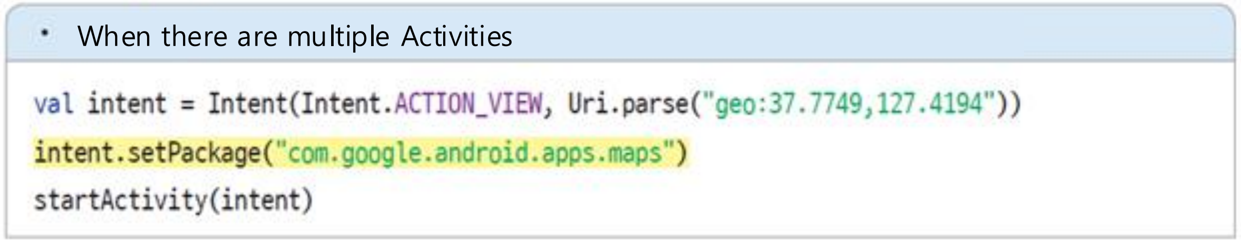
- When there are multiple Activities

```
val intent = Intent(Intent.ACTION_VIEW, Uri.parse("geo:37.7749,127.4194"))
startActivity(intent)
```



Intent (14)

- To launch an activity from a specific app, specify the package name of that app.



- When there are multiple Activities

```
val intent = Intent(Intent.ACTION_VIEW, Uri.parse("geo:37.7749,127.4194"))
intent.setPackage("com.google.android.apps.maps")
startActivity(intent)
```

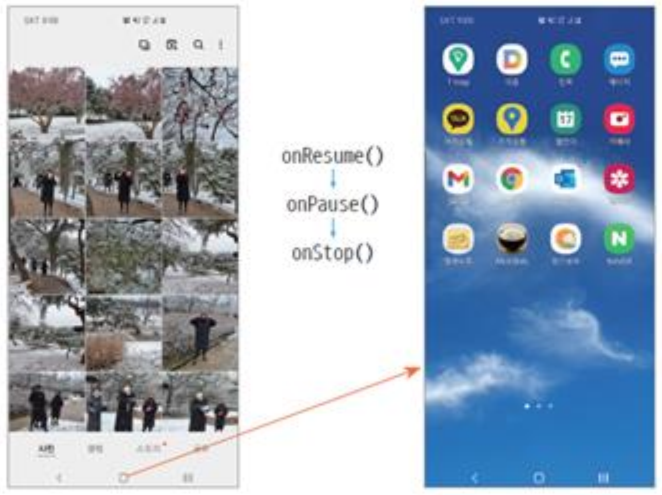
Activity Lifecycle (1)

- **Activity States**
  - The lifecycle refers to the process from when an activity is created until it is destroyed.
  - An activity can generally exist in three states:
    - Active: The activity is visible on the screen, and the user can interact with it.
    - Paused: The activity is visible but cannot receive user input events.
    - Inactive: The activity is not visible on the screen.

Activity Lifecycle (2)

- **Active State (Resumed)**
  - The activity is running, displayed on the screen, and can handle user input.
  - When first launched, the activity goes through the following callbacks:

    onCreate() → onStart() → onResume()
- **Paused State**
  - The onPause() method has been called.
  - The activity is still visible but has lost focus and cannot receive user input.
- **Inactive State (Stopped)**
  - The activity is not visible on the screen but has not been destroyed.
  - When transitioning from active to inactive, the following callbacks occur: onPause() → onStop()
- **When onDestroy() is called, it means the activity has been fully terminated.**

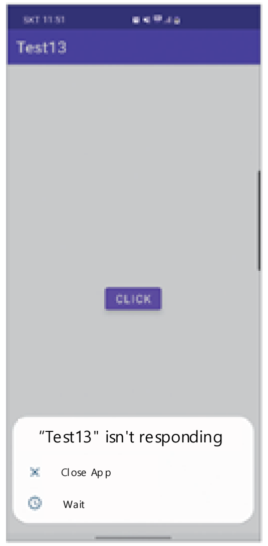


Activity Lifecycle (3)

- **Saving Activity State**
  - When an activity is terminated, its objects are destroyed and all data within the activity is lost.
  - Saving the state means preserving the activity’s data so that when it is restarted, the previous state can be restored.
  - For example, when the screen is rotated, the activity is destroyed and recreated — which causes all data to be reset unless the state has been properly saved.

Activity ANR problem and Coroutines (1)

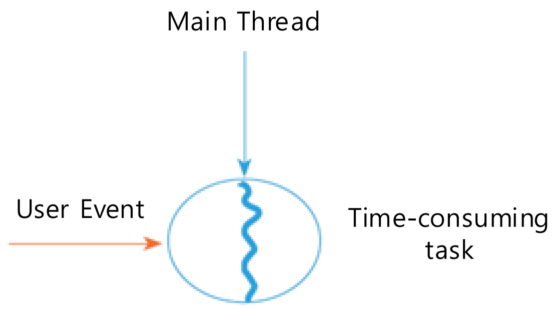
- **What is an ANR problem?**
  - ANR (Application Not Responding) occurs when an activity stops responding to user input.
  - The system runs the activity on the main thread, also known as the UI thread, which handles both the screen display and user interactions.



"Test13" isn't responding

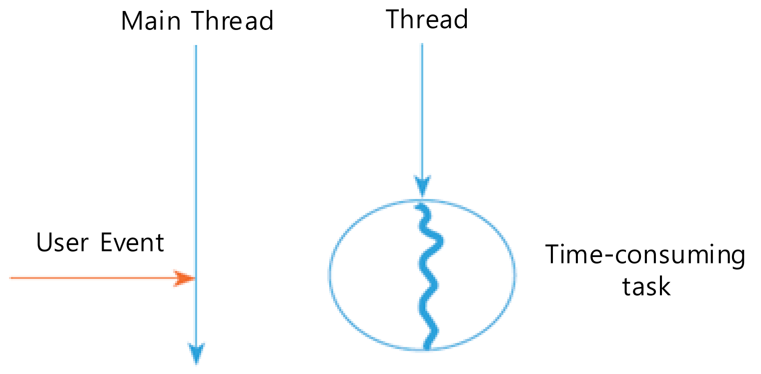
Close App

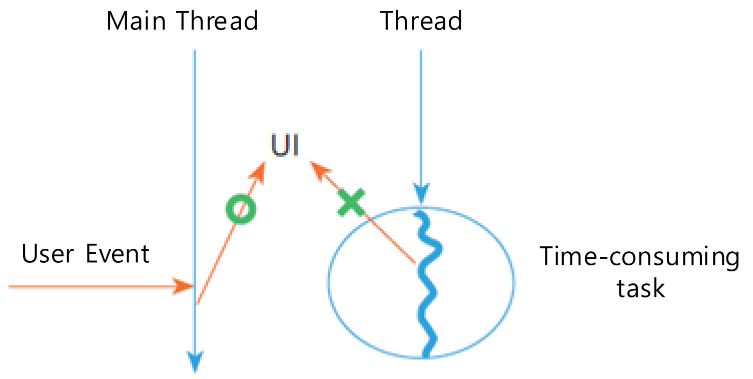
Wait



Activity ANR problem and Coroutines (2)

- **Solving the ANR problem**
  - The way to solve the ANR problem is to create a separate execution flow (developer thread) other than the main thread that runs the activity, and let it handle time-consuming tasks.
  - By doing so, the ANR error can be resolved, but another problem occurs — the screen cannot be updated.





```
android.view.ViewRootImpl$CalledFromWrongThreadException: Only the original thread that
created a view hierarchy can touch its views.
```

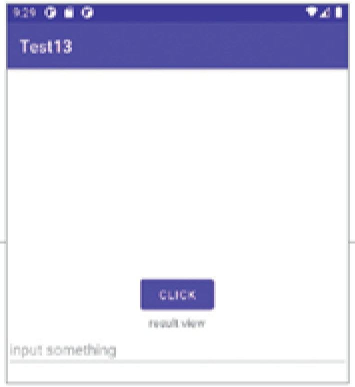
Activity ANR problem and Coroutines (3)

- **ANR Error**

- The example of time consuming task

```
var sum = 0L
var time = measureTimeMillis {
    for (i in 1..2_000_000_000) {
        sum += i
    }
}
Log.d("kkang","time : $time")
binding.resultView.text = "sum : $sum"
```

Execution Result



- **Using Coroutine in Android**

- Adding the coroutine dependency

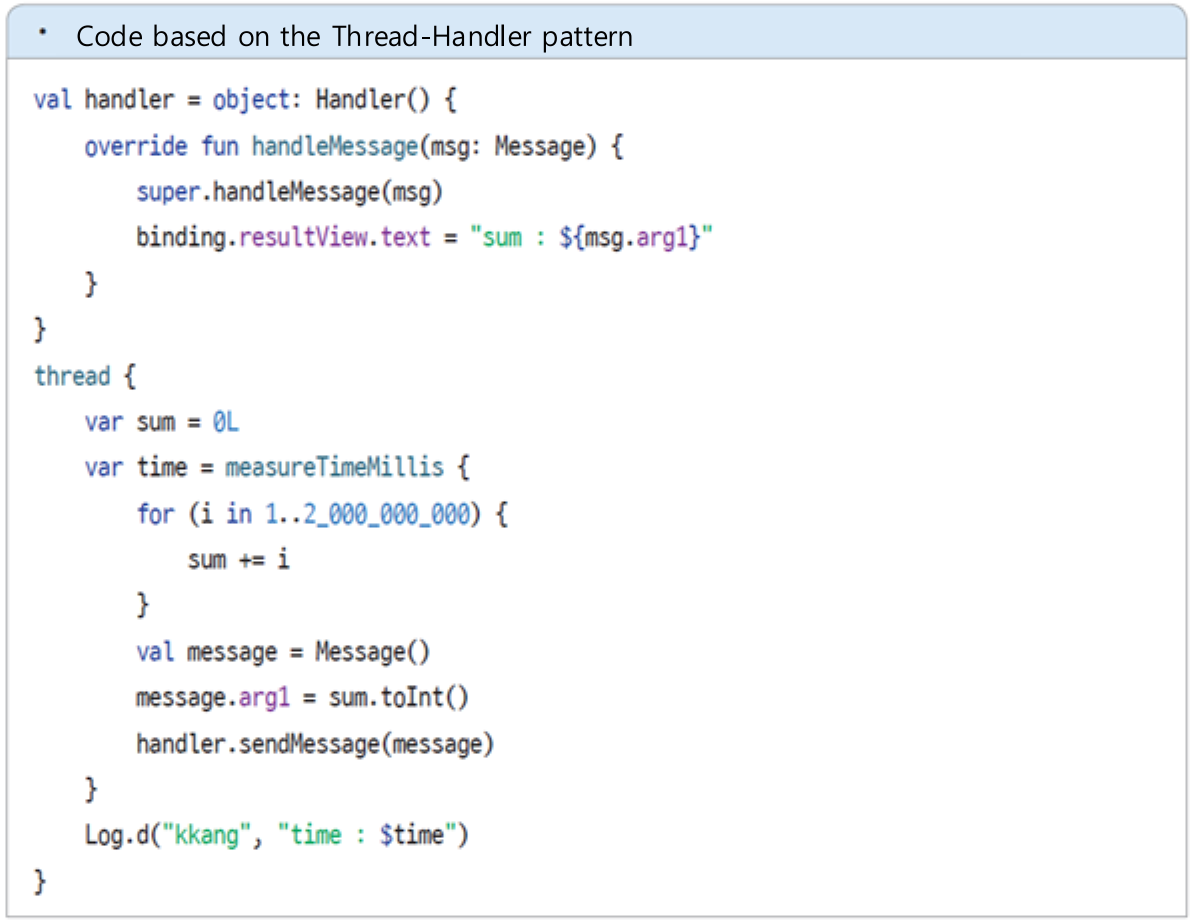
```
implementation("org.jetbrains.kotlinx:kotlinx-coroutines-android:1.7.3")
```

Activity ANR problem and Coroutines (4)

- **Solving ANR problem with Coroutines**
  - What is a Coroutine?
    - A coroutine is a lightweight thread for asynchronous processing.
    - It is a feature provided by the programming language.
  - Key Advantage of using Coroutines
    - It is lightweight.
    - It causes less memory leakage.
    - It supports various functions such as cancellation.
    - It is applied in many Jetpack libraries.

Activity ANR problem and Coroutines (5)

- **Fix ANR by Refactoring to a Thread–Handler Structure**



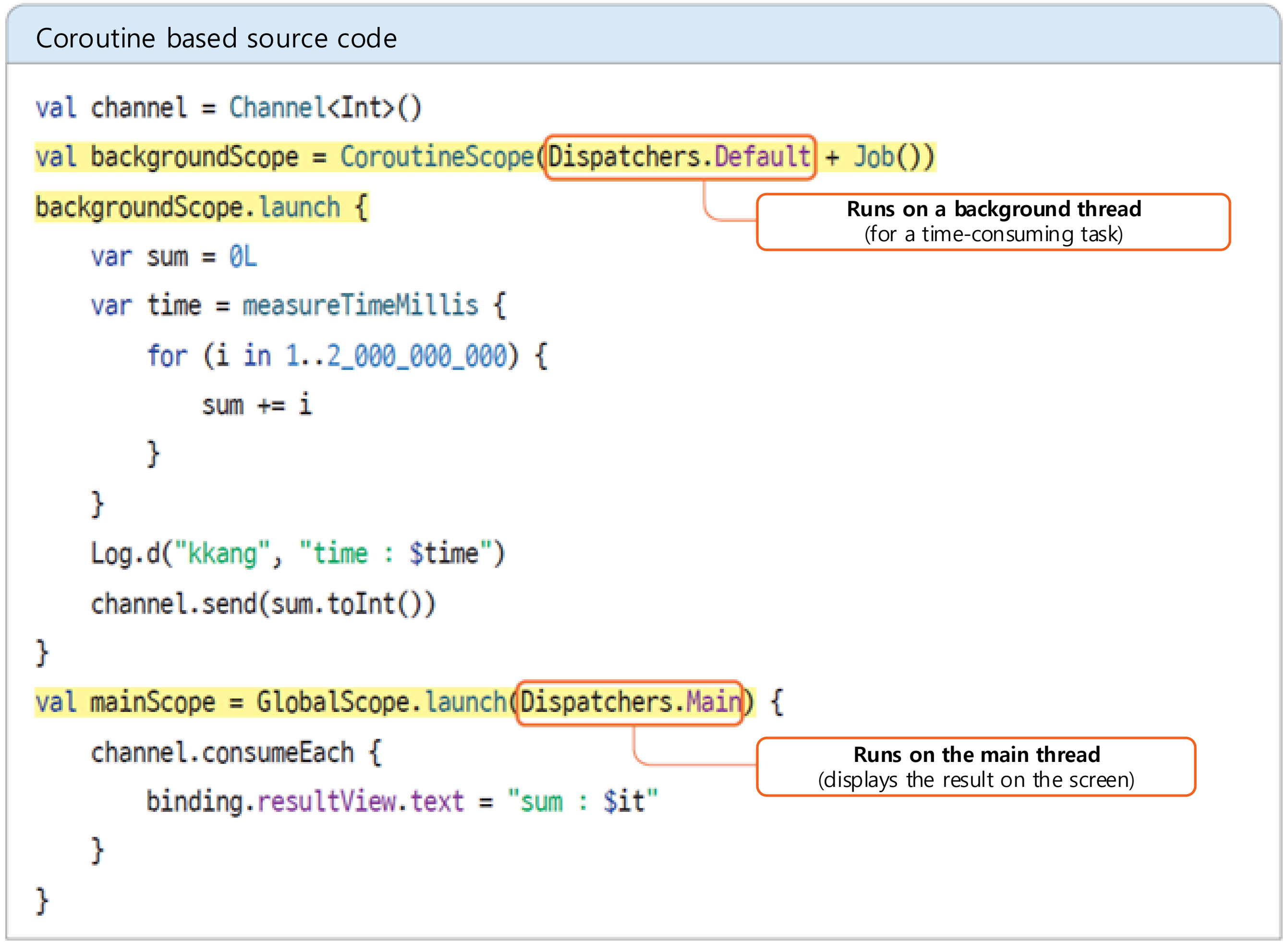
- Code based on the Thread-Handler pattern

```
val handler = object: Handler() {
    override fun handleMessage(msg: Message) {
        super.handleMessage(msg)
        binding.resultView.text = "sum : ${msg.arg1}"
    }
}
thread {
    var sum = 0L
    var time = measureTimeMillis {
        for (i in 1..2_000_000_000) {
            sum += i
        }
        val message = Message()
        message.arg1 = sum.toInt()
        handler.sendMessage(message)
    }
    Log.d("kkang", "time : $time")
}
```

Activity ANR problem and Coroutines (6)

- **Writing Code with Coroutine**
  - To run a coroutine, a scope must first be prepared.
  - Coroutines are executed within a scope.
  - A coroutine scope is an object of a class that implements CoroutineScope.
  - You can either implement it directly or use one of the scopes provided by Kotlin, such as GlobalScope, ActorScope, or ProducerScope.

Activity ANR problem and Coroutines (7)



Coroutine based source code

```
val channel = Channel<Int>()
val backgroundScope = CoroutineScope(Dispatchers.Default + Job())
backgroundScope.launch {
    var sum = 0L
    var time = measureTimeMillis {
        for (i in 1..2_000_000_000) {
            sum += i
        }
    }
    Log.d("kkang", "time : $time")
    channel.send(sum.toInt())
}
val mainScope = GlobalScope.launch(Dispatchers.Main) {
    channel.consumeEach {
        binding.resultView.text = "sum : $it"
    }
}
```

Runs on a background thread
(for a time-consuming task)

Runs on the main thread
(displays the result on the screen)

Broadcast Receiver (1)

- **Creating Broadcast Receiver**
  - Declare a class that inherits from BroadcastReceiver.
  - It has only one lifecycle function: onReceive().
  - Since the broadcast receiver is also a component, it must be registered in the manifest file.

- Creating broadcast receiver

```
class MyReceiver : BroadcastReceiver() {
    override fun onReceive(context: Context, intent: Intent) {
    }
}
```

- Adding broadcast receiver dependency

```
<receiver
    android:name=".MyReceiver"
    android:enabled="true"
    android:exported="true"></receiver>
```

Required properties

Broadcast Receiver (2)

- **Dynamic Registration and Unregistration**
  - Register dynamically when needed.
  - Use the registerReceiver() function to register with the system.
  - When no longer needed, unregister it using the unregisterReceiver() function.

- Create the receiver object

```
val receiver = object : BroadcastReceiver() {
    override fun onReceive(context: Context?, intent: Intent?) {
    }
}
```

- Dynamic Registration

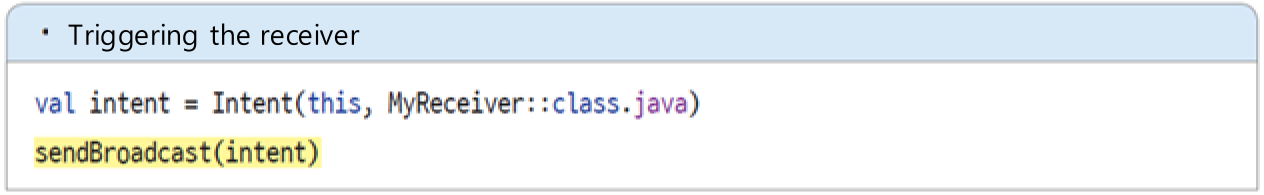
```
registerReceiver(receiver, IntentFilter("ACTION_RECEIVER"), RECEIVER_EXPORTED)
```

- Function to unregister the broadcast receiver

```
unregisterReceiver(receiver)
```

Broadcast Receiver (3)

- **Running a Broadcast Receiver**
  - An Intent is required to run a broadcast receiver.
  - A receiver registered in code using registerReceiver() can be executed with an implicit intent.
  - The intent that runs the receiver is delivered to the system using the sendBroadcast() function.



- Triggering the receiver

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

Broadcast Receiver (4)

Type	Number of Targets	Operation
Activity Intent	None	Error occurs
	1	Executes normally
	Multiple	One is executed by user selection
Receiver Intent	None	No error occurs
	1	Executes normally
	Multiple	All are executed

Checking System Status (1)

- **Boot Completed**
  - When the device finishes booting, the system sends an intent that contains the action string android.intent.action.BOOT_COMPLETED.

• Registering a Broadcast Receiver and Intent Filter

```
<receiver
    android:name=".MyReceiver"
    android:enabled="true"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED" />
    </intent-filter>
</receiver>
```

• Set authority

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

Checking System Status (2)

- **Screen ON / OFF**
  - A broadcast receiver that detects when the screen turns on or off cannot be executed if it is registered in the manifest.
  - Use the action strings android.intent.action.SCREEN_ON and android.intent.action.SCREEN_OFF.

• Screen on / off receiver

```
receiver = object : BroadcastReceiver() {
    override fun onReceive(context: Context?, intent: Intent?) {
        when (intent?.action) {
            Intent.ACTION_SCREEN_ON -> Log.d("kkang", "screen on")
            Intent.ACTION_SCREEN_OFF -> Log.d("kkang", "screen off")
        }
    }
}
```

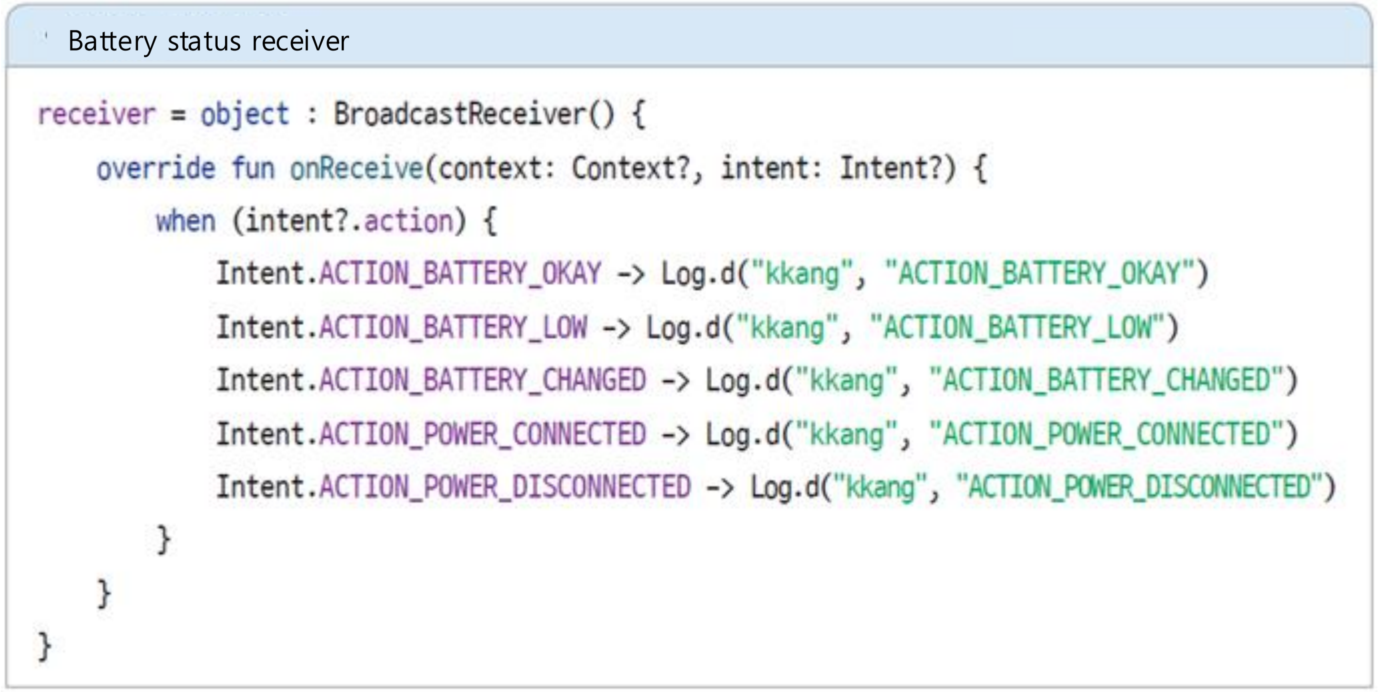
• Register receiver

```
val filter = IntentFilter(Intent.ACTION_SCREEN_ON).apply {
    addAction(Intent.ACTION_SCREEN_OFF)
}
registerReceiver(receiver, filter)
```

Checking System Status (3)

- **Battery Status**
  - When the battery status changes in the Android system, an intent is triggered with one of the following action strings:
    - BATTERY_LOW: Triggered when the battery becomes low
    - BATTERY_OKAY: Triggered when the battery returns to a normal state
    - BATTERY_CHANGED: Triggered when the charging status changes
    - ACTION_POWER_CONNECTED: Triggered when power is connected
    - ACTION_POWER_DISCONNECTED: Triggered when power is disconnected

Checking System Status (4)



Battery status receiver

```
receiver = object : BroadcastReceiver() {
    override fun onReceive(context: Context?, intent: Intent?) {
        when (intent?.action) {
            Intent.ACTION_BATTERY_OKAY -> Log.d("kkang", "ACTION_BATTERY_OKAY")
            Intent.ACTION_BATTERY_LOW -> Log.d("kkang", "ACTION_BATTERY_LOW")
            Intent.ACTION_BATTERY_CHANGED -> Log.d("kkang", "ACTION_BATTERY_CHANGED")
            Intent.ACTION_POWER_CONNECTED -> Log.d("kkang", "ACTION_POWER_CONNECTED")
            Intent.ACTION_POWER_DISCONNECTED -> Log.d("kkang", "ACTION_POWER_DISCONNECTED")
        }
    }
}
```

Checking System Status (5)

- **Battery Status**

• Getting battery status without a system Intent

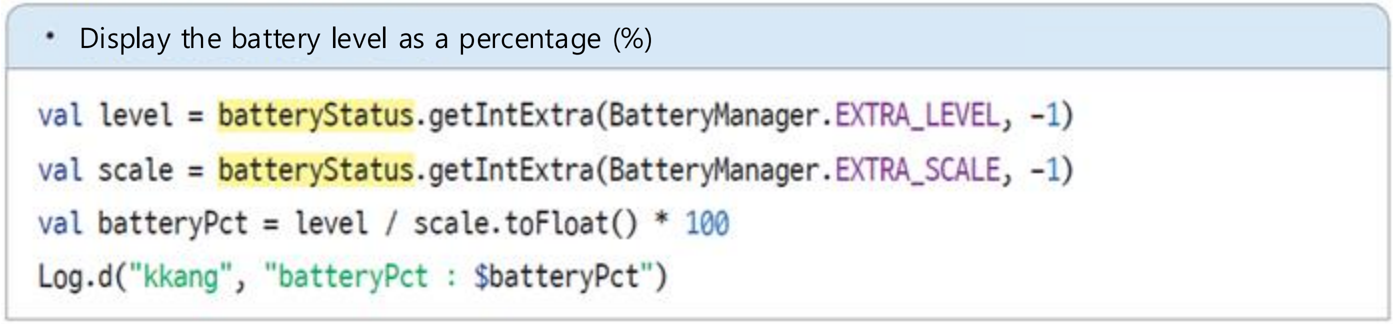
```
val intentFilter = IntentFilter(Intent.ACTION_BATTERY_CHANGED)
val batteryStatus = registerReceiver(null, intentFilter)
```

• Getting battery status using the Intent's extras

```
val status = batteryStatus!!.getIntExtra(BatteryManager.EXTRA_STATUS, -1)
if (status == BatteryManager.BATTERY_STATUS_CHARGING) {
     // If the device is charging
    val chargePlug = batteryStatus.getIntExtra(BatteryManager.EXTRA_PLUGGED, -1)
    when (chargePlug) {
        BatteryManager.BATTERY_PLUGGED_USB -> Log.d("kkang", "usb charge")
        BatteryManager.BATTERY_PLUGGED_AC -> Log.d("kkang", "ac charge")
    }
} else {
     // If the device is not charging
    Log.d("kkang", "not charging")
}
```

Checking System Status (6)

- When you want to know how much the battery is charged



- Display the battery level as a percentage (%)

```
val level = batteryStatus.getIntExtra(BatteryManager.EXTRA_LEVEL, -1)
val scale = batteryStatus.getIntExtra(BatteryManager.EXTRA_SCALE, -1)
val batteryPct = level / scale.toFloat() * 100
Log.d("kkang", "batteryPct : $batteryPct")
```

Q & A

aiclasscau@gmail.com