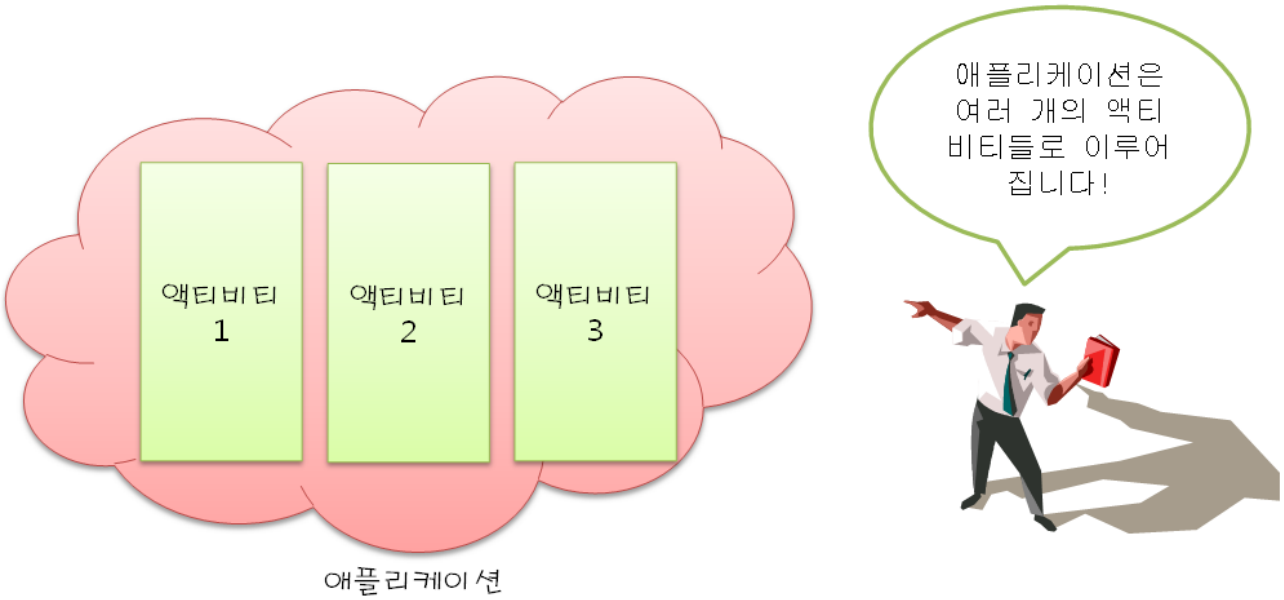
Activity LifeCycle

애플리케이션(Application)

- 한 개이상의 액티비티들로 구성
- 액티비티들은 애플리케이션 안에서 느슨하게 묶여 있음



Activity

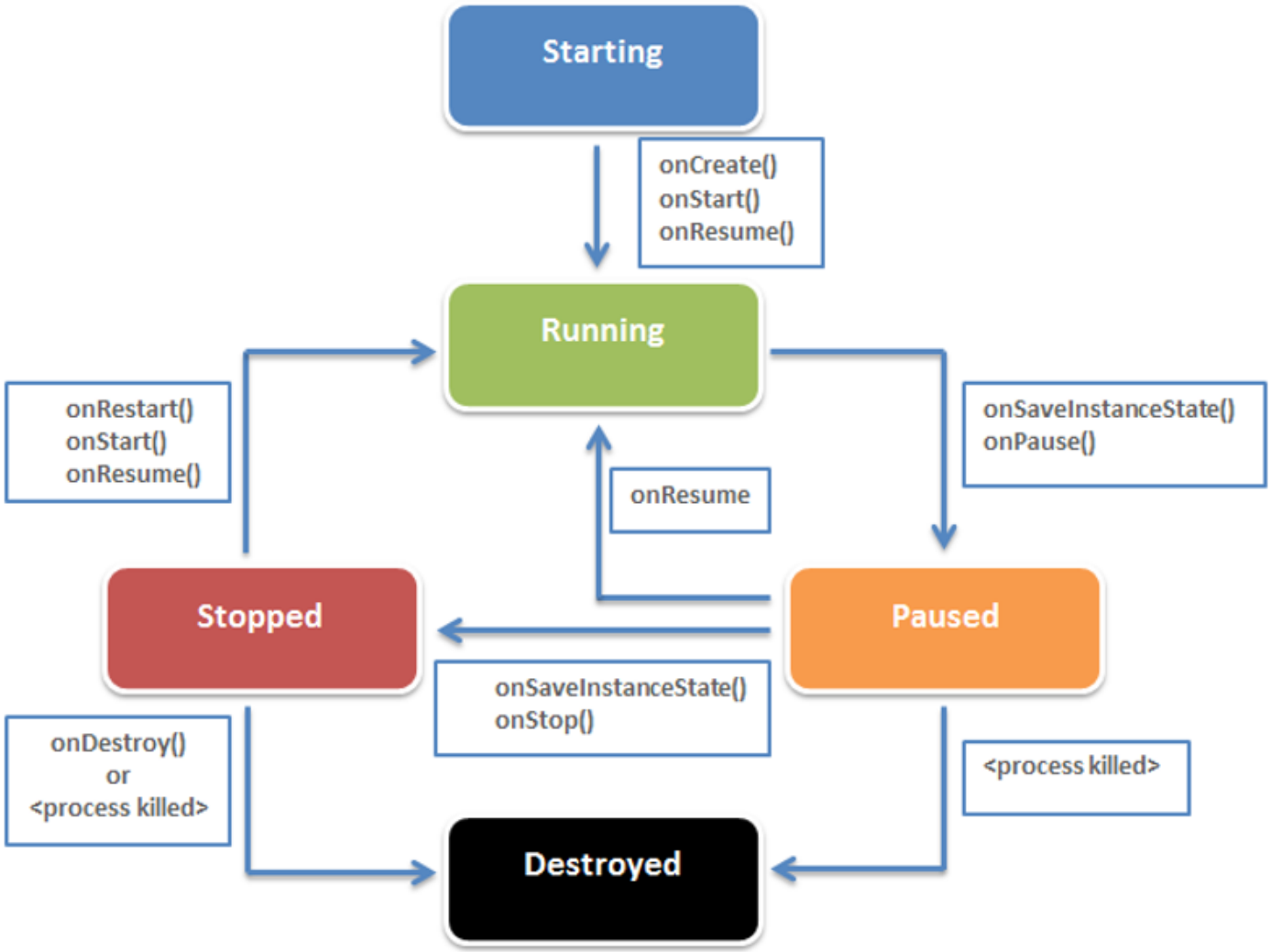
- 액티비티
  - 액티비티는 안드로이드 응용 프로그램을 구성하는 4가지 컴포넌트 중 하나로 가장 빈번히 사용되며 가장 중요한 요소
  - 화면 하나에 대응되며 입출력 기능이 없어 내부에 뷰나 뷰 그룹을 가짐
  - **setContentView** 메서드
    - 액티비티가 생성될 때마다 호출되며, 액티비티 안에 뷰를 배치하는 명령
  - 액티비티 하나는 독립된 기능을 수행하며, 서로 중첩되지 않음
  - 보안상의 이유로 응용 프로그램에 포함된 모든 액티비티는 반드시 매니페스트에 등록해야 함 (매니페스트에 등록되지 않은 액티비티는 존재하지 않는 것으로 취급)

```
        <activity android:name=".CallActivity" android:label="CallActivity" />
        <activity android:name=".SubActivity" android:label="SubActivity" />
    </application>
</manifest>
```

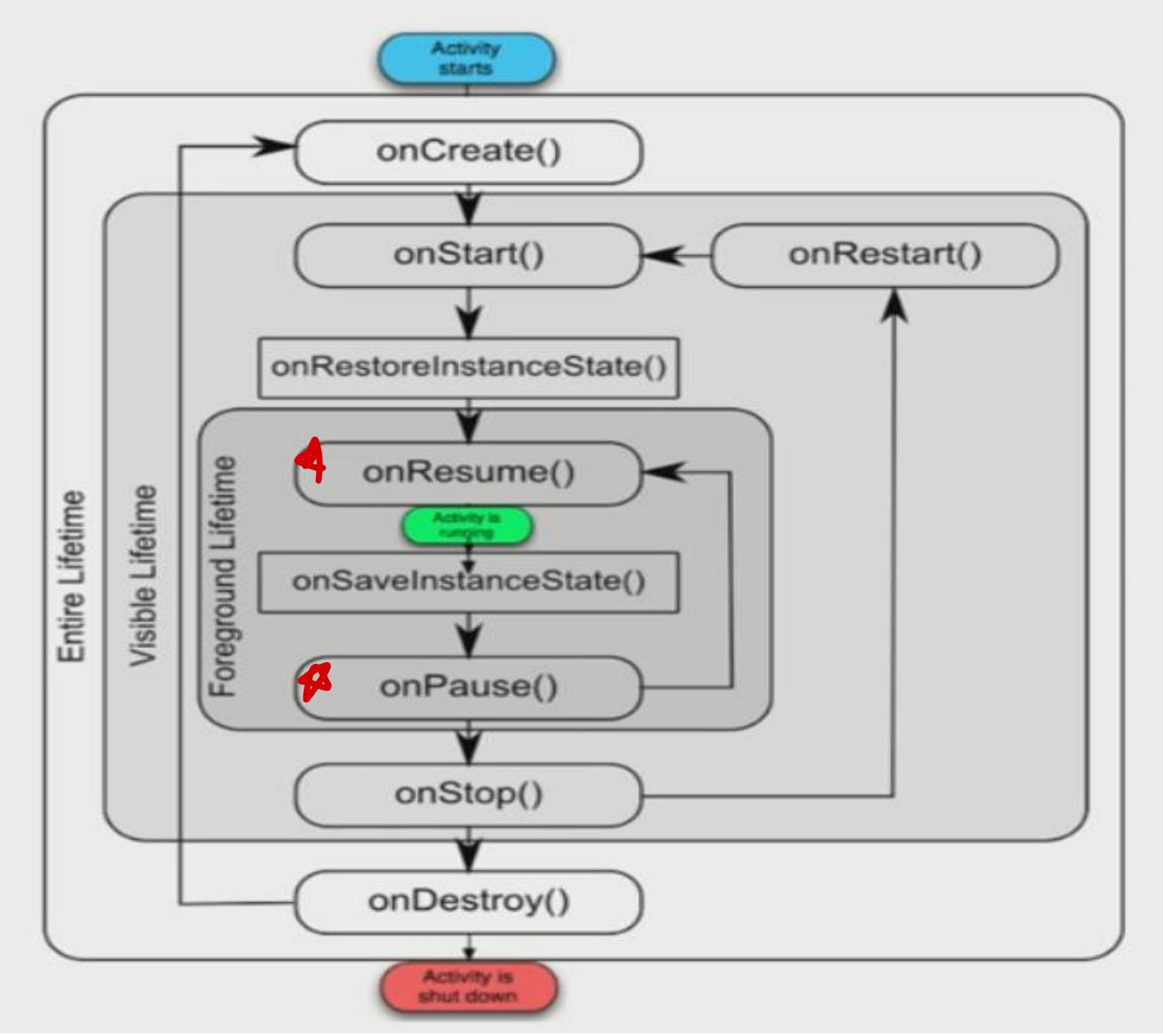
Activity State

- 액티비티는 여러 상태중의 하나의 상태를 나타냄
  - Starting : 로딩 중에 있으며, 아직 전체가 로딩되지는 않았음
  - Running : 로딩이 되었고, 스크린 상에 보여짐
  - Paused : 부분적으로 가려지거나, 포커스를 잃은 상태로, shut down은 되지 않은 상태
  - Stopped : 더 이상 활성화 상태는 아니지만, 메모리에는 있는 상태
  - Destroyed : Shut down되었고, 더 이상 메모리에 없는 상태
- 액티비티의 상태들 간의 전이는 이벤트로 표현됨
  - OnCreate, OnPause, OnResume, OnStop, OnDestroy, . . .

Activity Lifecycle

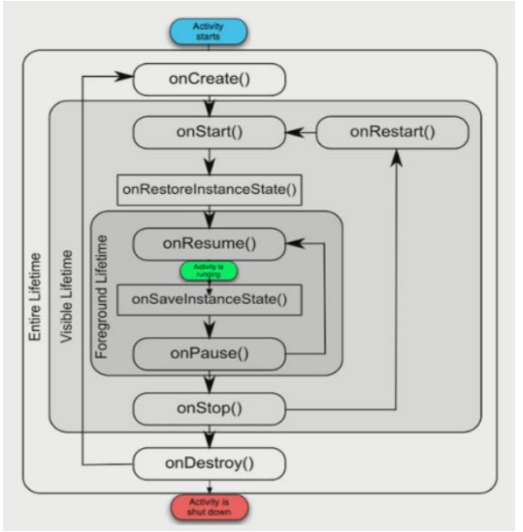


Activity State Diagram



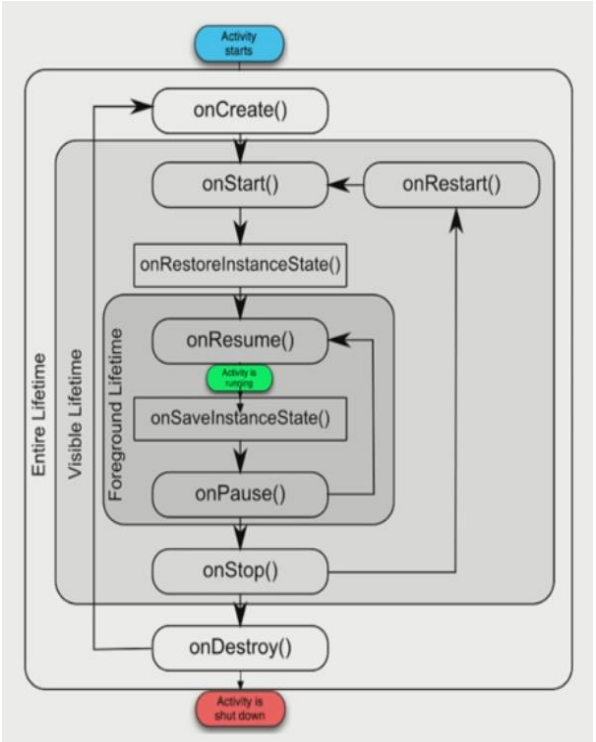
Activity State Method

- Callback(콜백) 메서드 : 지정된 상황에 호출되도록 약속된 메소드
- onCreate Method ➔ onStart()
  - 액티비티가 처음 생성될 때 호출
    - 이 곳에서 화면을 정의한 레이아웃 파일과 액티비티를 연결 해야 함
    - 액티비티 객체를 생성하고, 메뉴, 레이아웃, 이미지 같은 리소스를 로딩함
  - onCreate 이후에 액티비티가 존재하게 됨
  - 액티비티의 생성자처럼 생각하면 됨
- onStart Method ➔ onResume()
  - 액티비티가 사용자에게 표시되기 직전에 호출됨
- onResume Method ➔ onPause()
  - 액티비티가 시작되고, 사용자와 상호작용하기 직전에 호출
  - 사용자가 앱을 사용할 수 있음
  - 액티비티가 Pause 상태를 벗어나, 다시 Running 상태로 돌아올 때 호출됨
  - onPause에서 해제시켰던 리소스들을 초기화 함



Activity State Method

- onPause Method ➔ onResume() 또는 onStop()
  - 다른 액티비티를 표시하기 직전에 호출됨
    - 앱이 다시 시작할 경우를 대비해 변경 사항을 저장
  - 액티비티가 여전히 부분적으로 보여질 때
  - 임시적으로 중지, 또는 중지하는 중을 나타냄
    - CPU를 사용하는 애니메이션이나 액션들을 중지시킴
    - 저장되지 않는 변화(changes)들을 완료시킴(e.g. draft email)
    - 배터리를 소비하는 시스템 리소스들을 해제시킴
- onStop Method ➔ onRestart() 또는 onDestroy()
  - 액티비티가 더 이상 화면에 보여지지 않을 때 호출 됨
    - 사용자가 다른 액티비티를 시작할 때
    - 사용자가 앱에서 전화를 받는 경우
  - 앱을 여전히 실행되고 있지만, 액티비티는 아닐 경우
    - onStop 전에는 항상 onPause가 호출됨
    - onStop은 데이터베이스에 Commit하는 것과 같은 강력한 종료 작업을 수행함



Activity State Method

- onRestart Method
  - 액티비티가 stop 되었다가 다시 시작할 때 호출 됨
    - 첫 번째 시작만 제외하고 모두
  - 일반적으로 사용되는 것은 아님 (onResume을 더 많이 사용)
  - onStop이 닫은 모든 리소스를 다시 open함

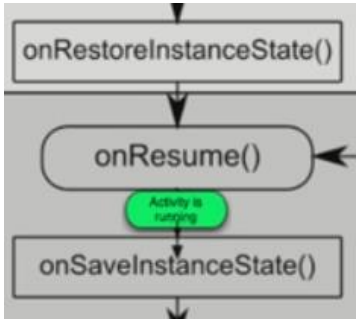
- onDestroy Method
  - 앱 전체가 종료되고, 메모리에서 unload될 때 호출 됨
    - 언제 호출될 지 정확하게 호출될지 알 수 없음
    - 시스템에서 앱에서 사용하는 메모리를 회수하려 할 때 호출할 수 있음
    - 예상 가능한 시기에 호출 되는 onPause 나 onStop을 더 많이 사용함

Activity State Method

- Activity 상태 저장 및 복원
  - 안드로이드 시스템은 액티비티의 상태를 저장해야 하는 경우 onSaveInstanceState() 콜백 메서드를 호출하고, 복구해야 하는 경우에는 onRestoreInstanceState() 콜백 메서드를 호출 함
    - Finish() 메서드를 강제적으로 호출하는 경우에는 저장과 복구를 위한 콜백 메서드는 호출되지 않음

```
<EditText
    ...
    android:saveEnabled="false"
    ... />
```

*동적저장 기능 제공되지만, 저장해 놓아야 하는 경우

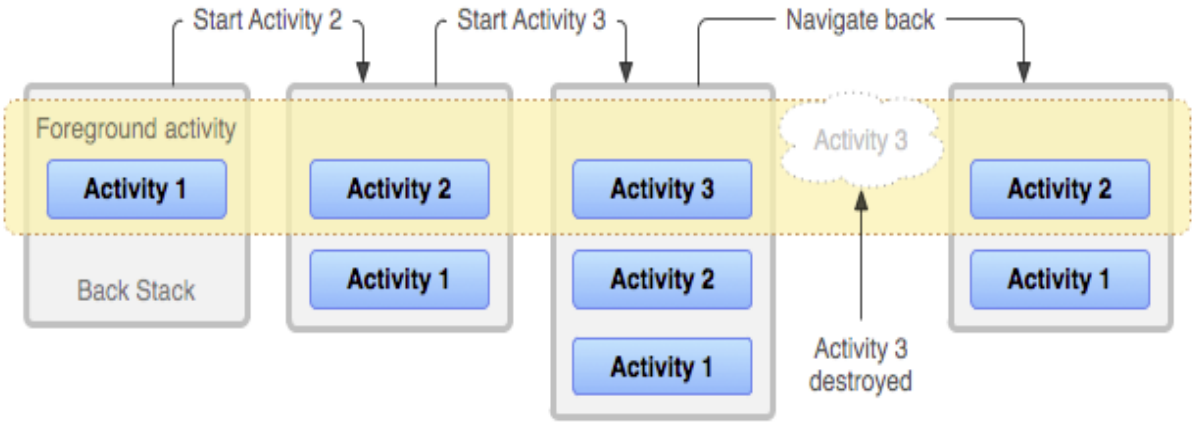


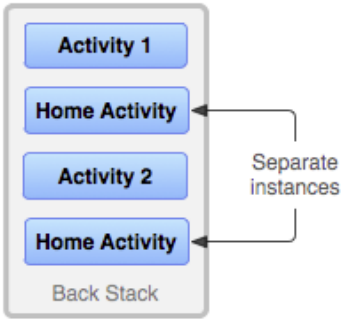

```
override fun onSaveInstanceState(outState: Bundle) {
    super.onSaveInstanceState(outState)
    val userText = editText.text
    outState.putCharSequence("savedText", userText)
}

override fun onRestoreInstanceState(savedInstanceState: Bundle) {
    super.onRestoreInstanceState(savedInstanceState)
    val userText = savedInstanceState.getCharSequence("savedText")
    editText.setText(userText)
}
```

Activity State Method

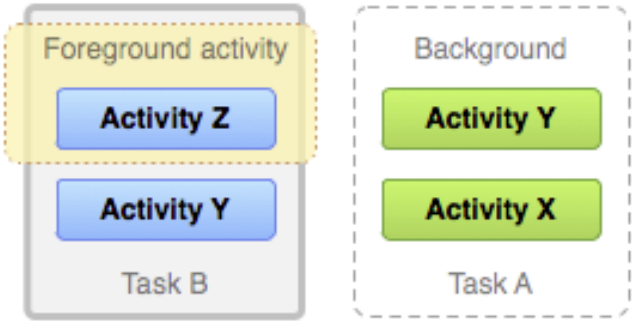
- Back stack을 이용한 화면 관리





*Back stack의 인스턴스는 정렬되지 않고, 새로운 인스턴스 저장

- Task의 back stack 유지



Activity State 출력

- LogCat system을 사용하여 앱의 상태 변화할 때 발생하는 로그 메시지를 출력
  - Android Studio에서 LogCat console에 나타남

```
val TAG = "MainActivity"


protected void onStart() {
    super.onStart();
    Log.v(TAG, "onStart");
}
```

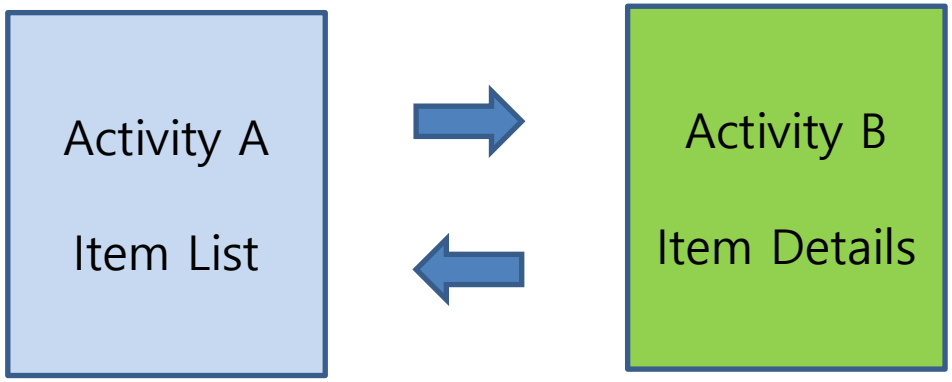
Log methods

- Log.v(“tag”,”message”)
  - Verbose : 개발중에만 사용하여 상세 정보 표시
- Log.i(“tag”,”message”)
  - Information : 일반 정보 표시
- Log.d(“tag”,”message”)
  - Debug : debug용 로그
- Log.w(“tag”,”message”)
  - Warning : 경고 표시
- Log.e(“tag”,”message”)
  - error : error용 로그

화면 전환

Activity

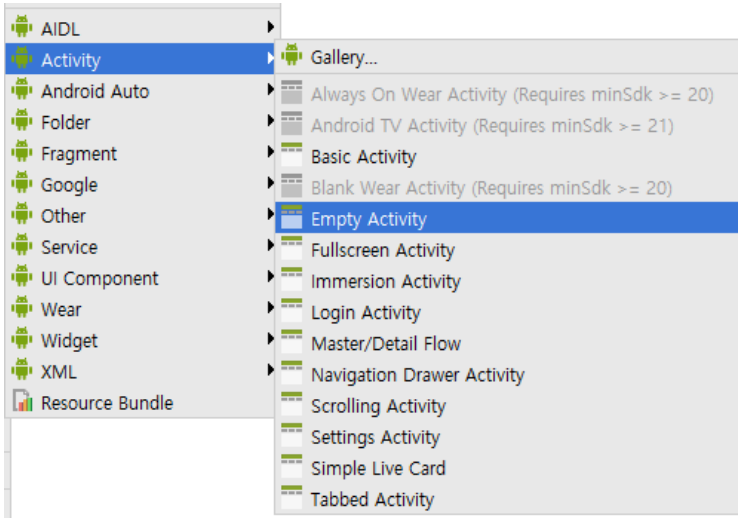
- 여러 개의 액티비티로 구성된 앱
  - 예) 연락처 앱에서, 연락처 리스트에서 클릭하면 자세한 정보를 보여주는 다른 액티비티로 전환됨

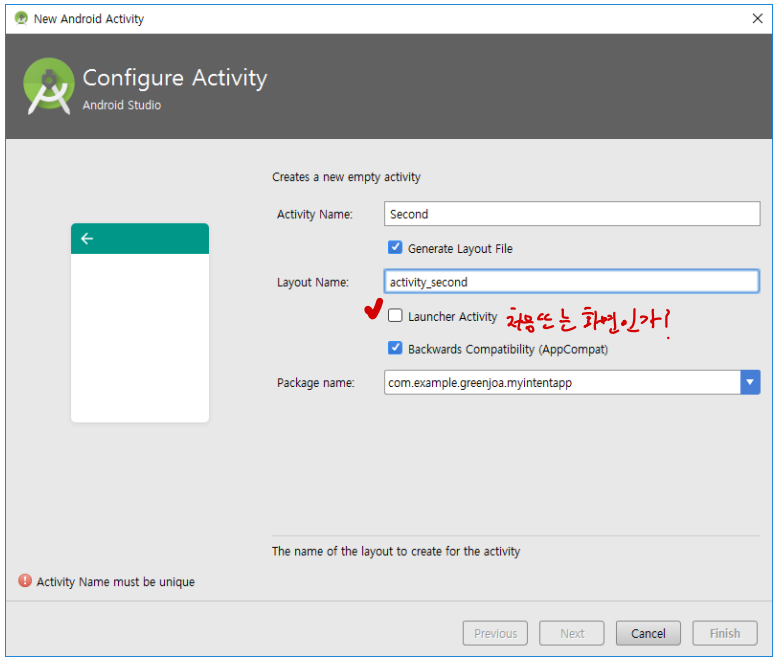


- 액티비티 A에서 액티비티 B를 실행시킬 수 있음
- 액티비티 A에서 B로 데이터를 전달할 수 있음
- 액티비티 B의 수행이 완료되면 A에게 데이터를 전달할 수 있음

Activity

- 액티비티 추가
  - 액티비티 추가 예제
    - 메인 액티비티 ➔ 두번째 액티비티 – **Second.java**, **activity_second.xml**





Manifest에 등록

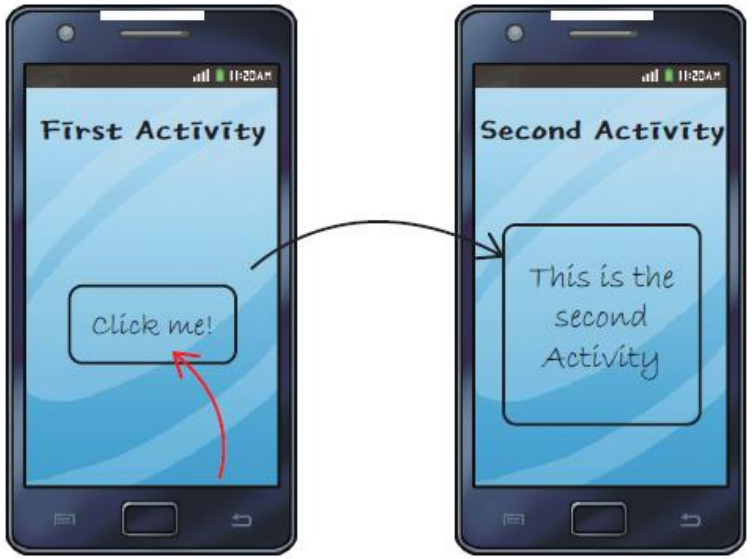
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.greenjoa.myintentapp">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity"> 액티비티 태그
            <intent-filter> 인텐트 필터 태그
                <action android:name="android.intent.action.MAIN" /> 액션 태그
                <category android:name="android.intent.category.LAUNCHER" />
                                                                    카테고리 태그
            </intent-filter>
        </activity>
        <activity android:name=".Second"></activity>
    </application>

</manifest>
```

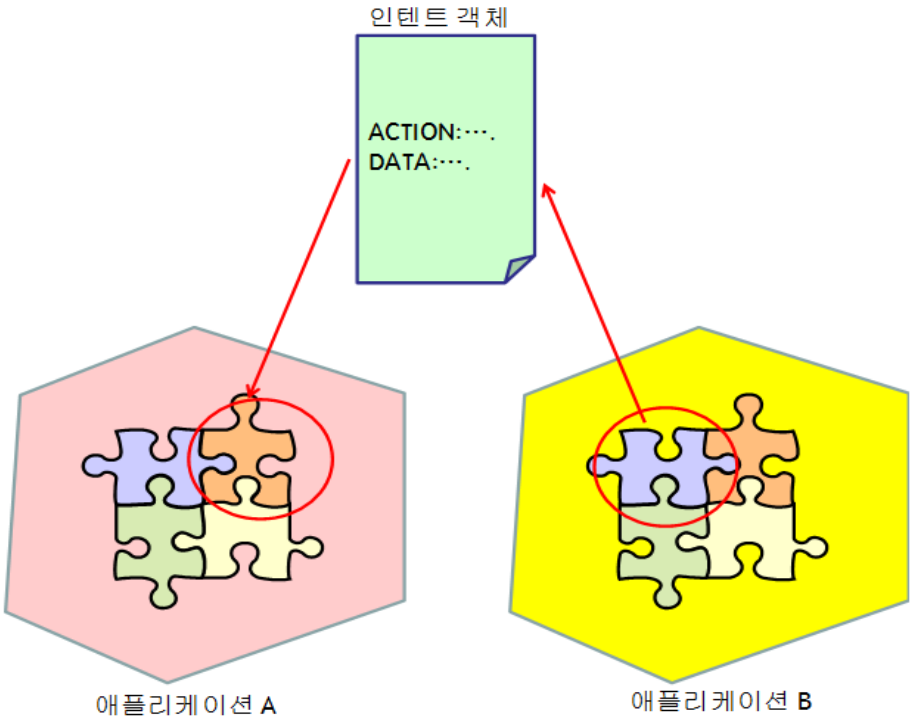
Intent

- 각각의 화면은 별도의 액티비티로 구현됨
- 하나의 액티비티에서 다른 액티비티로 전환하려면 어떻게 해야 하는가?



Intent

- 안드로이드의 컴포넌트끼리 통신하기 위한 메시지 시스템
- 다른 액티비티를 시작하려면 액티비티의 실행에 필요한 여러가지 정보들을 보내주는 역할을 수행함



Intent의 종류

- 명시적 인텐트(explicit intent)
  - 인텐트를 수행할 class를 명시적으로 지정

```
Intent(Context packageContext, Class<?> cls)
```

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

- 암시적 인텐트(implicit intent)
  - 인텐트의 Action 메시지를 통해 함축적으로 수행할 class가 결정

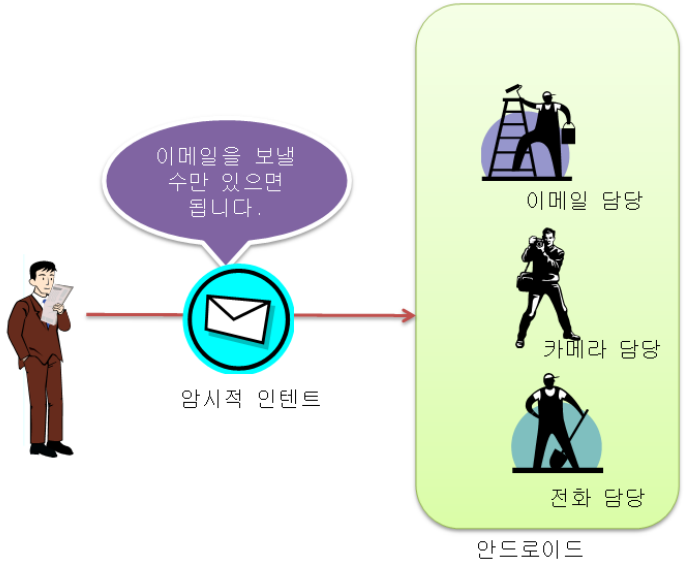
```
Intent(String action)
Intent(String action, Uri uri)
```

```
val intent = Intent(Intent.ACTION_DIAL);
val intent = Intent(Intent.ACTION_CALL);
val intent = Intent(Intent.ACTION_CALL, Uri.parse("tel:"+"114"));
startActivity(intent);
```

Implicit Intent

Implicit Intent

- 어떤 작업을 하기를 원하지만 그 작업을 담당하는 컴포넌트의 이름을 정확하게 모르는 경우에 사용




```
val uri = Uri.parse("mailto:"+editText.text.toString())
val intent = Intent(Intent.ACTION_SENDTO, uri)
startActivity(intent)
```

Implicit Intent

- 액션의 종류

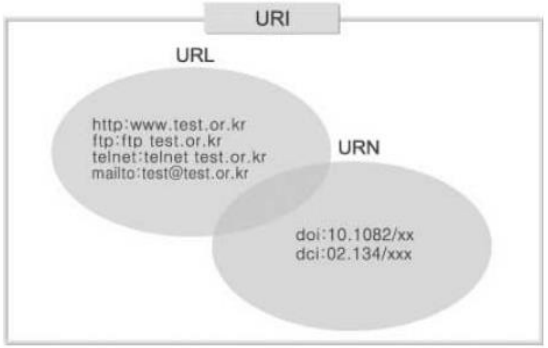
상수	타겟 컴포넌트	액션
ACTIN_VIEW	액티비티	데이터를 사용자에게 표시한다.
ACTION_EDIT	액티비티	사용자가 편집할 수 있는 데이터를 표시한다.
ACTION_MAIN	액티비티	태스크의 초기 액티비티로 설정한다.
ACTION_CALL	액티비티	전화 통화를 시작한다.
ACTION_SYNC	액티비티	데이터 동기화를 수행한다.
ACTION_DIAL	액티비티	전화 번호를 누르는 화면을 표시한다.

Implicit Intent

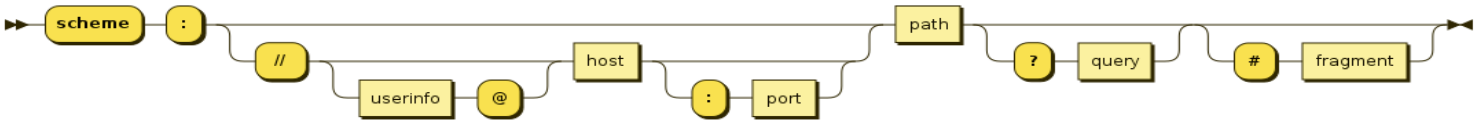
- 암시적 인텐트 예

액션 / 데이터

- ACTION_VIEW content://contacts/people/1 – 1번 연락처 정보를 표시한다.
- ACTION_DIAL content://contacts/people/1 – 1번 연락처로 전화걸기 화면을 표시한다.
- ACTION_VIEW tel:0101234567 – 0101234567번 전화번호로 전화걸기 화면을 표시한다.
- ACTION_DIAL tel:0101234567 -- 0101234567번 전화번호로 전화걸기 화면을 표시한다.
- ACTION_EDIT content://contacts/people/1 -- 1번 연락처 정보를 편집한다.
- ACTION_VIEW content://contacts/people/ -- 연락처 리스트를 표시한다.



- URI (Uniform Resource Identifier)
  - 인터넷상의 정보 자원에 대한 식별체계
    - 인터넷 주소체계(URL) + 고유이름 체계(URN)을 총칭하는 개념
  - 문서, 이미지, 음악파일, 동영상 등 다양한 정보자원에 대한 유일성을 부여하고, 식별을 가능하게 하는 관리 체계



View Binding

- 뷰와 상호작용하는 코드를 쉽게 작성하는 기능
  - 레이아웃에 ID가 있는 모든 뷰의 직접 참조가 가능하게 함
  - 뷰 객체를 참조하기 위해 findViewByID 사용했던 것 대체 함
    - Null safety : 유효하지 않은 뷰 ID로 인한 null 포인터 발생 위험 없음
    - Type safety : XML 파일의 뷰 타입과 일치하지 않는 위험 없음
- 모듈별 뷰 바인딩 설정 (build.gradle)

Android Studio 3.6~4.0

```
android {
    ...
    viewBinding {
        enabled true
    }
}
```

Android Studio 4.0이상

```
android {
    ...
    buildFeatures {
        viewBinding true
    }
}
```

```
private lateinit var binding: ActivityMainBinding

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

퍼미션

시스템 퍼미션

- 시스템에서 보호하고 있는 특정 기능을 위한 퍼미션
  - 이 기능을 사용하는 앱은 <uses-permission>을 반드시 선언해 주어야 함.

https://developer.android.com/reference/android/Manifest.permission

- 퍼미션 구조

https://developer.android.com/guide/topics/manifest/permission-element?hl=ko

```
<permission android:description="string resource"
            android:icon="drawable resource"
            android:label="string resource"
            android:name="string"
            android:permissionGroup="string"
            android:protectionLevel=["normal" | "dangerous" |
                                     "signature" | ...] />
```

시스템 퍼미션

- 퍼미션의 Protection Level <android:protectionLevel>
  - Normal
    - 위험성이 낮은 권한으로, 사용자의 명시적인 승인을 요구하지 않고 설치시 권한을 요청하는 애플리케이션에 권한을 자동 부여
  - Dangerous
    - 위험성이 높은 권한으로, 사용자에게 명시적으로 표시하고, 확인을 요청함
  - Signature
    - 동일한 키로 서명된 앱만 실행

시스템 퍼미션

- 대표적인 예

CALL_PHONE

Added in API level 1

```
public static final String CALL_PHONE
```

Allows an application to initiate a phone call without going through the Dialer user interface for the user to confirm the call.

Protection level: dangerous ..... 사용자에게 권한 부여할 지 물어봄

Constant Value: "android.permission.CALL_PHONE"

- AndroidManifest.xml 파일에 퍼미션 요청

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

Permission Check하기

- 안드로이드 6.0(Marshmallow)이후의 권한 설정 방법
  - 기기 제어(카메라, 블루투스, GPS, 파일읽기 등)할 수 있는 권한은 사용자의 동의를 받아야만 사용할 수 있고, **사용자는 언제든지 권한 사용을 취소할 수 있음**
  - 사용 절차
    - Manifest 파일에 권한 정보 등록하기

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

    - 권한 체크
      - 만족하는 경우
        - » 정상 기능 수행
      - 만족하지 않는 경우
        - » 권한 요청
          - 권한 요청 결과에 따른 제어

Permission Check하기

- checkSelfPermission 함수
  - 권한을 만족하는지 체크하는 함수
  - 반환값 : PERMISSION_GRANTED / PERMISSION_DENIED
- shouldShowRequestPermissionRationale 함수
  - 권한 승인을 명시적으로 거부한 경우 True 반환

```
when{
   ActivityCompat.checkSelfPermission(this,
      Manifest.permission.CALL_PHONE)==PackageManager.PERMISSION_GRANTED->{
        // 권한이 승인 되어 있을 때 수행할 내역
   }

   ActivityCompat.shouldShowRequestPermissionRationale(this, Manifest.permission.CALL_PHONE)->{
        // 권한을 명시적으로 거부했을 때 수행할 내역
   }

   else->{
        // 처음 수행할 때 수행할 내역
   }
}
```

Permission Check하기

- requestPermissions 함수
  - 권한 요청하는 함수
  - 권한 설정 창이 띄워짐
    - 요청 퍼미션, 상태코드 값(결과 확인때 이용)

```
ActivityCompat.requestPermissions(this,
    arrayOf(Manifest.permission.CALL_PHONE),
    CALL_REQUEST)
```

Permission Check하기

- onRequestPermissionsResult
  - 사용자의 퍼미션 허용이 끝나면 자동으로 호출되는 함수
  - 사용자의 퍼미션 허용 결과를 확인하는 함수

```
override fun onRequestPermissionsResult(requestCode: Int,
                                        permissions: Array<String>, grantResults: IntArray) {

    super.onRequestPermissionsResult(requestCode, permissions, grantResults)

    when(requestCode){
        CALL_REQUEST -> {
            if (grantResults[0] == PackageManager.PERMISSION_GRANTED){
              // 퍼미션 동의했을 때 할 일
            } else {
              // 퍼미션 동의하지 않았을 때 할일
              // 앱종료 finish();
            }
        }
    }
}
```

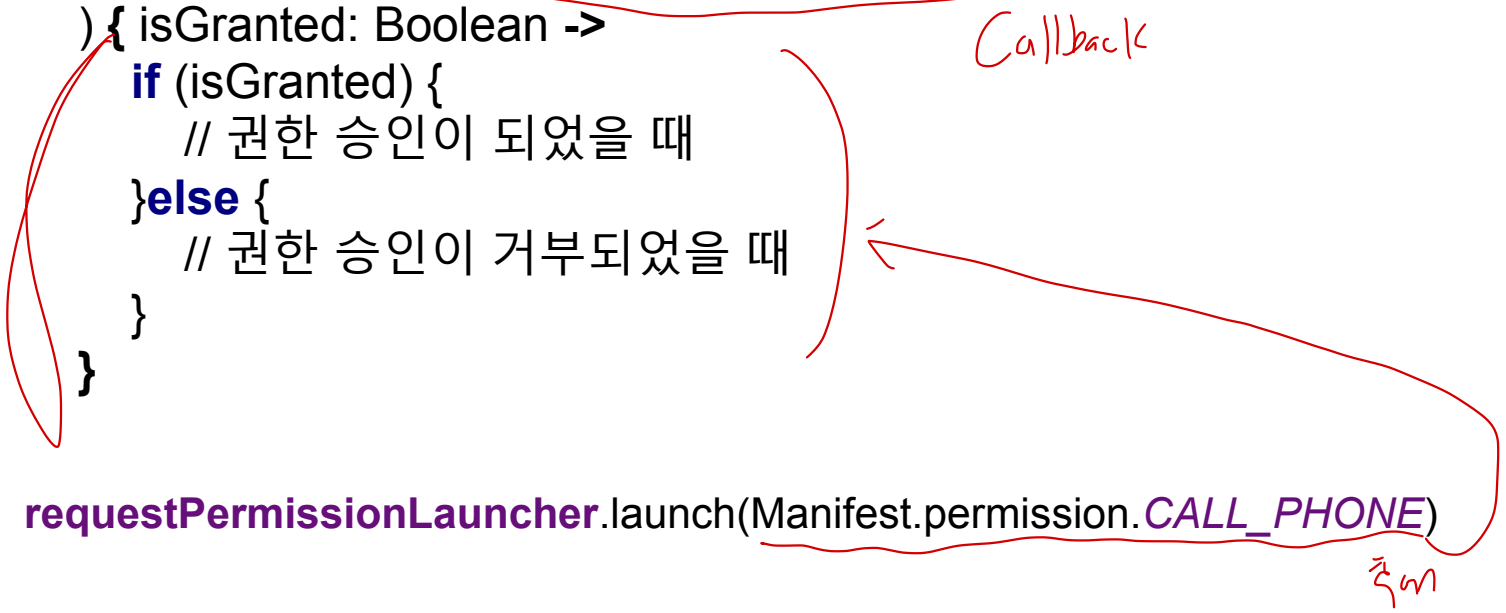
Permission Check하기

- androidx.activity library에서 제공하는 런타임 권한 체크
  - 기존보다 간결한 방법으로 권한 요청 가능
  - registerForActivityResult 함수로 CallBack 함수 등록

- 단일 권한 요청

```
val requestPermissionLauncher =  registerForActivityResult(
        ActivityResultContracts.RequestPermission()
    ) { isGranted: Boolean ->
        if (isGranted) {
            // 권한 승인이 되었을 때
        }else {
            // 권한 승인이 거부되었을 때
        }
    }

requestPermissionLauncher.launch(Manifest.permission.CALL_PHONE)
```



Permission Check하기

- 복수 권한 요청
  - RequestMultiplePermissions()

```
val permissions = arrayOf(
    Manifest.permission.CALL_PHONE,
    Manifest.permission.CAMERA)

val requestMultiplePermissionsLauncher =
    registerForActivityResult(ActivityResultContracts.RequestMultiplePermissions()) { map ->
     // map (permission, Boolean)

}

requestMultiplePermissionsLauncher.launch(permissions)
```

Explicit Intent

Explicit Intent

- Activity간 메시지 전달할 때 사용
  - 명확하게 Activity가 정해져 있는 경우
  - Intent 보내는 쪽

```
val i = Intent(context, IntentB::class.java);
startActivity(i);
```

  - Intent 받는 쪽

```
public class IntentB : Activity(){
override fun onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);

val i = intent; // getIntent()
…
}
…
}
```

Explicit Intent

- Intent를 통한 Activity간 메시지 전달(Explicit Intent)
  - **Intent를 호출할 때 메시지를 실어보내기**
  - Intent를 보내는 쪽

```
public class A : Activity() {
....
val i = Intent(this, B::class.java);
i.putExtra("int", 5);
i.putExtra("string", "hello");
startActivity(i);
}
```

  - Intent 받는 쪽

```
public class B : Activity() {
public void onCreate(Bundle savedInstanceState) {
val i = intent;
val number = i.getIntExtra("int", -1);
                    //-1: default값. pair되는 값이 없을 경우 default로 -1 return.
val string = i.getStringExtra("string");
}
}
```

Explicit Intent

- Intent를 통한 Activity간 메시지 전달(Explicit Intent)
  - **Intent를 통해 메시지 전달 받기**

```
val activityResultLauncher = registerForActivityResult(
    ActivityResultContracts.StartActivityForResult()){it:ActivityResult!
        if(it.resultCode==Activity.RESULT_OK){       }
}
```

```
val intent = Intent(this, IntentB.class);
activityResultLauncher.launch(intent)
```

IntentB 클래스

```
{
...
val result = Intent();
result.putExtra("result", "greet");
setResult(RESULT_OK,result);
finish()
...
}
```

```
public class IntentA : Activity() {
...
val i = Intent(this, IntentB.class);
startActivityForResult(i, requestCode);
...
}

override fun onActivityResult(requestCode: Int,
                              resultCode: Int, data: Intent?)
{ // requestCode : 어느 Intent에 대해서 넘어온 것인지 확인가능.
  if(requestCode==123){
      if(resultCode == RESULT_OK){ }
      else{ }
  }
```

r) deprecated

Explicit Intent

- 객체를 넘기고 싶은 경우는 Serializable 기능 이용
  - 직렬화 기능
    - 순서대로 저장되고, 순서대로 보내도록 하는 것
    - 주고 받고자 하는 클래스에 Serializable 상속
    - 받을 경우, getSerializableExtra 함수 이용

* 클래스

```
data class MyData(val name:String, val addr:String) : Serializable
```

* 보내는 클래스

```
val intent = Intent(this, SecondActivity::class.java)
intent.putExtra("greenjoa", MyData("greenjoa", "seoul"))
activityResultLauncher.launch(intent)
```

* 받는 클래스

```
val data = intent.getSerializableExtra("greenjoa") as MyData
```

파일 출력하기

- **내부 저장공간에 파일 생성하기**

* 파일 저장하기 : /data/data/패키지이름/files/파일명

```
val output = PrintStream(
             openFileOutput("test.txt", Context.MODE_PRIVATE))
output.println("Hello")
...
output.close()
```

* MODE_PRIVATE : 자기 앱 내에서만 사용, 디폴트
* MODE_APPEND : 기존 파일에 추가

* 파일 읽어오기

```
val scan = Scanner(openFileInput("test.txt"))
```

수고하셨습니다.