안드로이드 기본 컴포넌트
Activity

- http://developer.android.com

목차

액티비티 (Activity)

- 하나의 화면을 제어하는 핵심 컴포넌트

주요구성요소 4가지 中 하나

- Activity 구성 요소
  - Activity에 표시할 View: 보통 XML로 작성
  - Activity 클래스 (상속받아 사용)

- Activity 추가 방법
  - 직접 추가
    - 액티비티에 표시할 View(Layout) XML 파일 작성 (코딩 시 생략)
    - Activity 클래스 구현 (AppCompatActivity 상속, onCreate() 구현)
    - AndroidManifest.xml 에 Activity 정보 등록
  - 메뉴 사용
    - 프로젝트명에서 우클릭 → [New] → [Activity] → 용도별 액티비티를 선택하여 추가 (기본은 Empty View Activity)

인텐트 (Intent) 1

- 안드로이드 컴포넌트 사이의 통신 수단 ≒ 메시지

택배 상자

- 인텐트의 유형
  - Explicit Intent(명시적 인텐트) → 실행 대상을 명확히 지정
  - Implicit Intent(묵시적 인텐트) → 적절한 대상의 조건만을 지정하여 호출

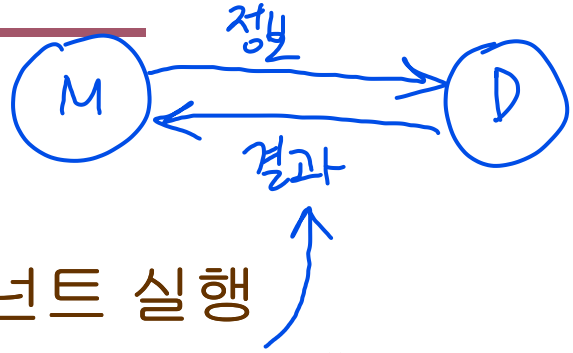
- 인텐트 주요 생성자

```
• Intent (packageContext : Context, targetClass: Class<T>)
• Intent (action : String [, uri : URI ])
```

- 인텐트에 저장할 수 있는 정보
  - Action, Data, Type, Category, Component, Extras, Flags
  - 각 정보에 대해 setter/getter 존재

★ 넣고싶은 데이터 아무거나

인텐트 (Intent) 2



인텐트의 사용

- 컴포넌트 실행 및 필요 정보를 저장
- 인텐트를 OS에 전달하여 필요한 컴포넌트 실행
  - Activity 실행: startActivity() 또는 startActivityForResult()
  - Service실행 startService(), BR실행 sendBroadcast()

- 예: 버튼 클릭 시 DetailActivity 실행 (명시적 인텐트)

```
binding.button.setOnClickListener{ it: View!
    val intent = Intent(this, DetailActivity::class.java)
    startActivity(intent)
}
```

- 실행할 Activity 클래스 정보 → 명시적 인텐트
- Intent 를 사용하는 Context
- 현재 Activity 가 Context 역할
- 새로운 Activity 실행 메소드

묵시적(Implicit) 인텐트의 사용

- 외부 패키지의 안드로이드 컴포넌트 호출
  - 예: 내 앱에서 외부에 있는 카메라앱 호출하기

- 특정 기능에 대한 정보를 인텐트에 저장한 후 호출
  - 안드로이드 운영체제가 인텐트를 토대로 적합한 컴포넌트를 검색 후 실행 (각 앱의 AndroidManifest 참조)

- 사용 예
  - 지정한 웹주소 확인

• 실행 동작의 유형 → Action 필드

• 실행에 필요한 추가 정보 → Data 필드

```
val intent = Intent(Intent.ACTION_VIEW, Uri.parse("https://www.google.com"))
startActivity(intent)
```

  - 지정한 번호로 전화 기능 실행

```
val intent = Intent(Intent.ACTION_DIAL, Uri.parse("tel:012-3456-7890"))
startActivity(intent)
```

• 참조: https://developer.android.com/guide/components/intents-filters

Intent를 이용한 정보 전달 1

Intent 의 Extra 필드 이용

- 개발자 임의로 전달하려는 데이터가 있을 때 사용
- Bundle 타입의 키와 값의 조합으로 자료 저장 가능
- 대부분의 자료형과 객체도 저장 가능

오버로딩 되어있음

- putExtra (key: String, value: Int) : Intent
- putExtra (key: String, value: String) : Intent
- …
- putExtra (key: String, value: Serializable) : Intent

- DTO 같은 객체를 Intent로 전달할 때 → 해당 객체는 Serializable 인터페이스를 구현한 클래스의 객체여야만 함
- ArrayList 등 컬렉션 클래스는 Serializable 를 구현하고 있음

- Intent 에 저장한 Extra 필드 값 읽기
  - 자료형 별로 메소드 존재

- getIntExtra (key: String, defaultValue: Int) : Int
- getStringExtra (key: String) : String
- …
- getSerializableExtra(key : String) : Serializable

…

Intent를 이용한 정보 전달 2 – String의 경우

MainActivity

```
val intent = Intent(this, DetailActivity::class.java)
// intent 의 extra 에 "subject" 로 "mobile software" 저장
intent.putExtra("subject", "mobile software")
startActivity(intent)
```

- DetailActivity 를 대상으로 명시적 인텐트 생성
- Intent의 Extra에 전달할 데이터 추가
- Activity 시작

Intent

DetailActivity

```
// "subject"라는 키로 저장한 String 을 intent에서 확인
val data = intent.getStringExtra("subject")
Log.d(TAG, "Subject: $data")
```

- getIntent()를 사용하여 전달받은 intent 확인 가능
- 코틀린의 경우 바로 intent 사용
- 저장한 데이터의 자료형에 따라 메소드 사용

Intent를 이용한 정보 전달 3 – DTO의 경우

덕여자대학교

내가만든 클래스추가

기본적으로는 안 되지만 있으면 가능 (없으면 불가)

- DTO 등 객체를 Intent로 전달하고자 할 경우 → Serializable 인터페이스 등을 구현하여야 함

```
class FoodDto(val photo: Int, val food: String, var count: Int) : Serializable
```

- 별도로 구현하여야 할 함수는 없음

MainActivity

```
val dto = FoodDto(R.mipmap.ic_launcher,"치킨", 10)
val intent = Intent(this, DetailActivity::class.java)
intent.putExtra("food", dto)
startActivity(intent)
```

DetailActivity

- "food" 라는 key로 저장한 객체 확인
- FoodDto 로 형변환
- as 형변환대상클래스

```
val data = intent.getSerializableExtra("food") as FoodDto
// val data = intent.getSerializableExtra("food", FoodDto::class.java)
Log.d(TAG, "음식: ${data.food}")
```

- API 33 에서 변경

Intent를 이용한 정보 전달 4 – 결과값 반환

호출 액티비티

- startActivityForResult() 를 사용하여 액티비티 호출
- 피호출 액티비티의 반환 결과를 onActivityResult()로 확인

```
val DETAIL_ACTIVITY_CODE = 100
```

• 호출 구분을 위한 상수 선언

• MainActivity

```
binding.button.setOnClickListener{ it: View!
    val intent = Intent(this, DetailActivity::class.java)
    startActivityForResult(intent, DETAIL_ACTIVITY_CODE)
}
```

• 호출부분
• startActivityForResult() 사용
• intent와 호출코드 값 전달

• 반환 결과 확인 부분
• 호출한 액티비티를 종료하고 결과값을 전달받으면 자동 호출

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    when (requestCode) {
        DETAIL_ACTIVITY_CODE -> {
            if (resultCode == RESULT_OK) {
                val result = data?.getStringExtra("result_data")
                Toast.makeText(this, "Result: $result", Toast.LENGTH_SHORT).show()
            }
        }
    }
}
```

• RESULT_OK
• RESULT_CANCELED

• 결과 값을 담고 있는 인텐트

Intent를 이용한 정보 전달 5 – 결과값 반환

피호출 액티비티

- setResult()에서 결과상태와 결과를 저장할 인텐트 생성
  - RESULT_OK 또는 RESULT_CANCELED
- 결과 설정 후 finish()를 호출하여 액티비티 종료

• DetailActivity

```
binding.btnOk.setOnClickListener{ it: View!
    val resultIntent = Intent()
    resultIntent.putExtra("result_data", "DetailActivity returns data!")
    setResult(RESULT_OK, resultIntent)
    finish()
}

binding.btnCancel.setOnClickListener { it: View!
    setResult(RESULT_CANCELED)
    finish()
}
```

• 정상 반환일 경우
• RESULT_OK // Activity 상수
• 결과를 담고 있는 인텐트 설정

신호만 보내도 됨!

• 취소일 경우
• RESULT_CANCELED // Activity 상수

• 현재 액티비티 종료

Activity 실행 1

새로운 액티비티를 실행할 경우

MainActivity
(호출 Activity)

이벤트 발생 시
(버튼 클릭 등)

1. 인텐트 생성

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

[optional] 데이터를 전달할 경우

```
// intent.putExtra ("key", "hello")
```

2. Activity 호출

```
startActivity ( intent )
```

DetailActivity
(피호출 Activity)

```
onCreate(...) {

 [optional] 데이터를 전달받았을 경우
 // val intent = getIntent();
 // String일 경우
 // val data = intent.getStringExtra("key");

...
}
```

intent 대신 쓰면 됨!

Activity 실행 2

호출 Activity 에서 결과 Intent를 받고자 할 경우

MainActivity
(호출 Activity)

이벤트 발생 시
(버튼 클릭 등)

1. 인텐트 생성

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

[optional] 데이터를 전달할 경우

```
// intent.putExtra ("key", "hello")
```

2. Activity 호출

```
startActivityForResult (intent, REQ );
```

3. Activity 결과 수신

```
onActivityResult ( REQ, RES, resultIntent );
```

DetailActivity
(피호출 Activity)

```
onCreate(...) {
```

[optional] 데이터를 전달받았을 경우

```
// val intent = getIntent();
// String일 경우
// val data = intent.getStringExtra("key");
}
```

결과 생성 및 종료 시
(버튼 클릭 등)

1. 결과 저장

```
val resultIntent = Intent();
resultIntent.putExtra ("key", "HI!");
setResult ( RES, resultIntent );
```

2. 액티비티 종료

```
finish();
```

이전 액티비티로 탐색 추가

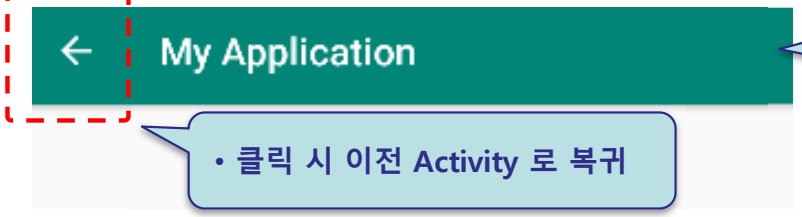
앱 호출 계층(Main : Detail)에 따른 이전 화면 가기

AndroidManifest 편집

```
<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>

<activity
    android:name=".DetailActivity"
    android:exported="false" >
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MainActivity" />
</activity>
```

- 앱에서 하나의 액티비티가 이 부분을 갖고 있어야 함
- action: 주요 진입 지점 의미
- category: 사용자가 이 액티비티를 시작할 수 있도록 함

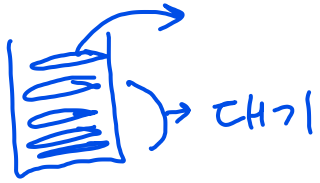


- 액션바가 안나올 경우 res/values/themes/themes.xml 중 noActionBar 부분 제거
- 클릭 시 이전 Activity 로 복귀

Activity Lifecycle

- Activity는 실행 시 여러 상태를 거치게 됨
  - 생성 → 준비 → 사용 → 대기 …
  - 생성부터 최종적으로 파괴까지의 상태 변화를 Activity의 생명주기(Life Cycle)이라고 지칭

- Activity Stack

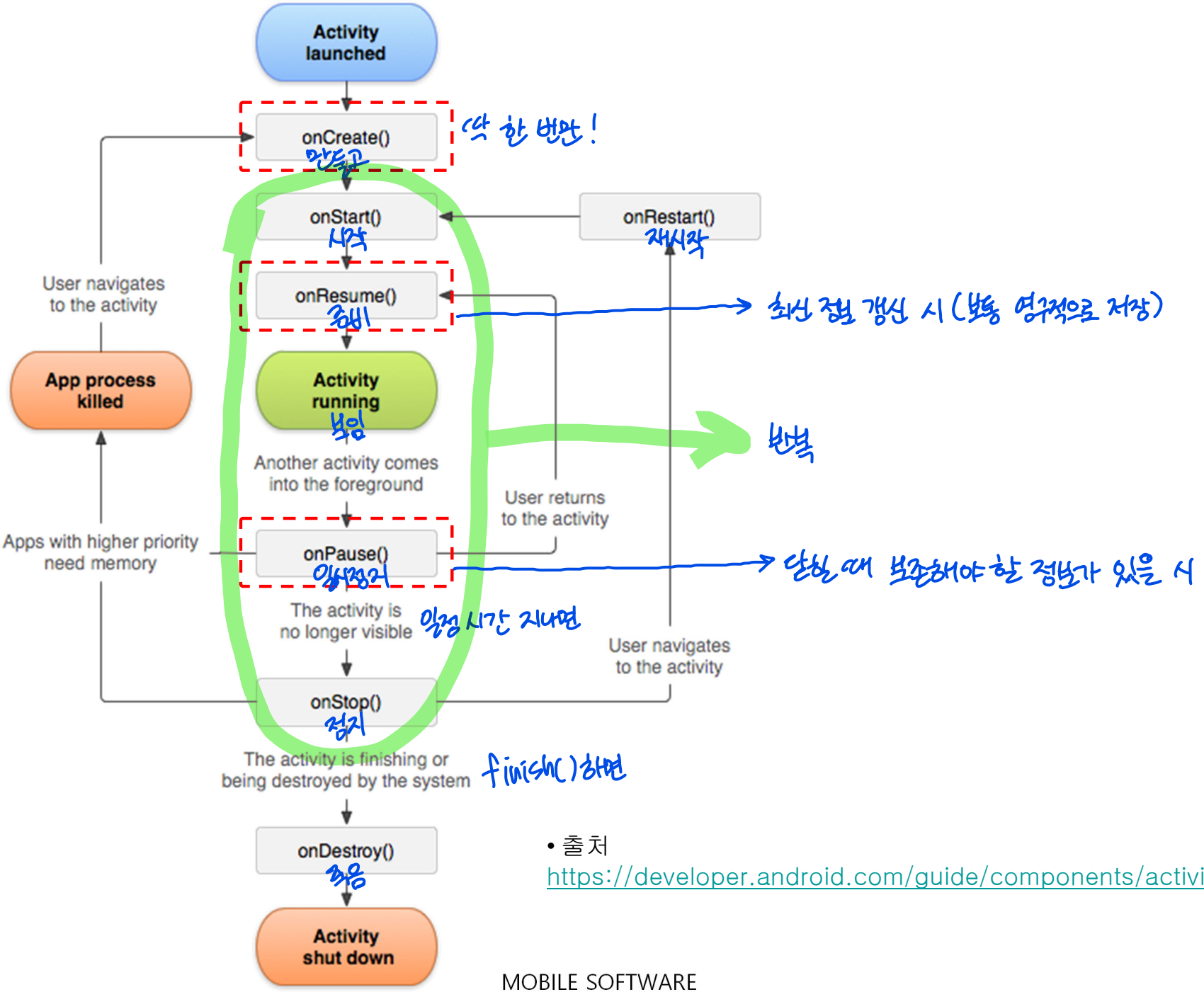


Activity를 보관해야 할 ccq!

  - 실행 중인 Activity를 차례대로 저장
  - Stack 상의 Activity 상태
    - 실행: 사용자가 Activity 를 화면에서 사용하는 상태 top(O)
    - 일시 정지: 사용자에게 보이지만 대기중인 상태 top(X)
    - 정지: Activity가 완전히 가려져 사용자에게 보이지 않는 상태

- Activity의 실행 상태에 따라 Activity가 갖고 있는 Life Cycle 관련 메소드가 자동으로 실행

Activity Lifecycle에 따른 메소드 호출



• 출처
https://developer.android.com/guide/components/activities.html

Activity Lifecycle 핵심 메소드

onCreate()

최소 1회만!

- Activity 화면 생성 등 생성을 위해 필요한 작업 수행
- 필수 구현

onResume()

- Activity가 사용자와 상호작용 하기 직전(보여지기 직전)에 호출
- 화면 내용 갱신 등의 작업 수행

onPause()

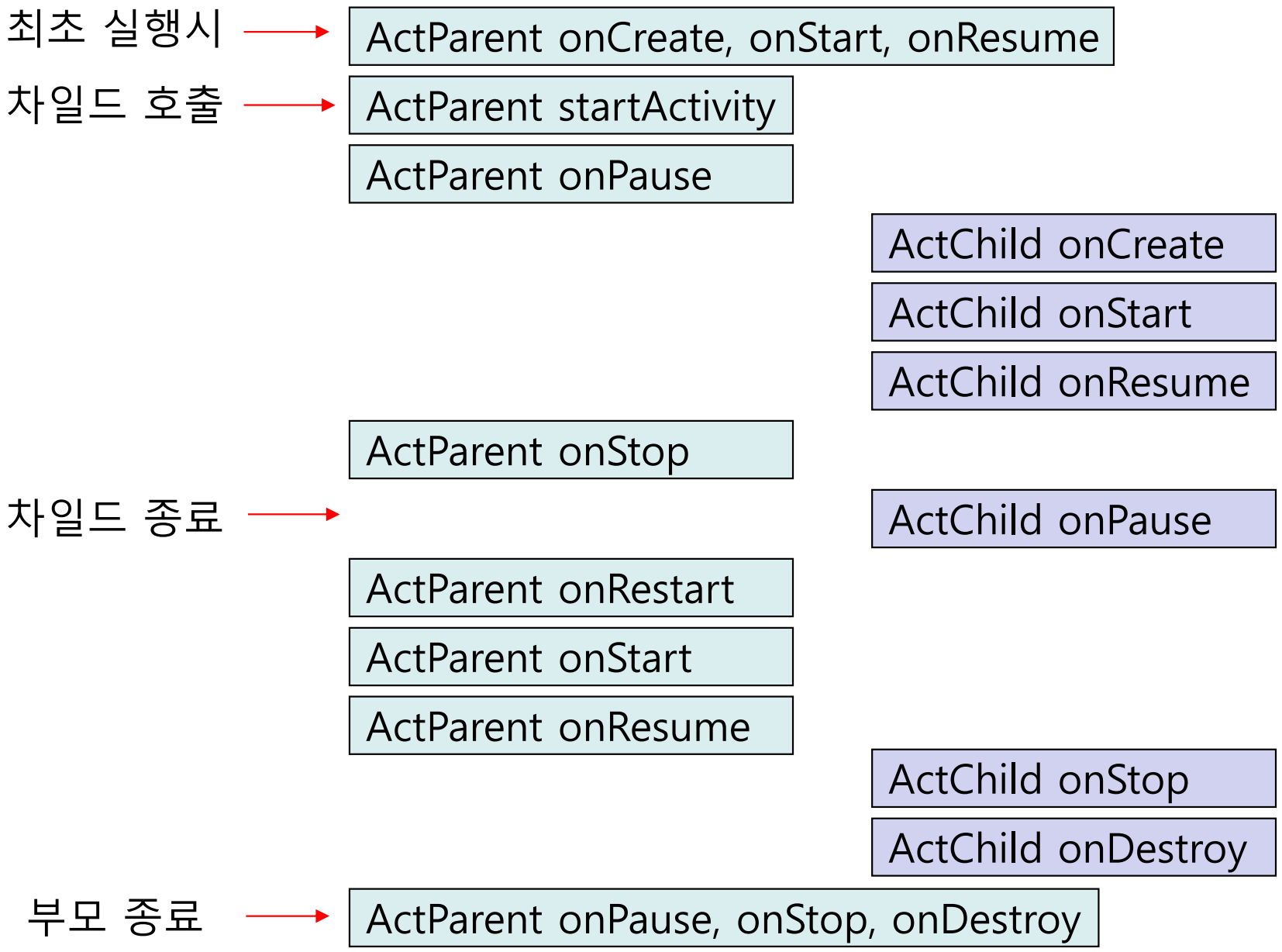
- 다른 Activity에 가려지거나 Activity 전체가 보여지지 않을 때 호출
- 앱 실행 중에 보존해야 할 정보가 있을 경우 이 메소드에서 백업 수행

Activity Lifecycle 메소드 요약

메서드	설명	완료 후 중단 가능?	다음
onCreate()	액티비티가 처음 생성되었을 때 호출됩니다. 이곳에서 일반적인 정적 설정을 모두 수행해야 합니다. 즉 뷰 생성, 목록에 데이터 바인딩하기 등을 말합니다. 이 메서드에는 액티비티의 이전 상태가 캡처된 경우 해당 상태를 포함한 번들 객체가 전달됩니다(이 문서 나중에 나오는 액티비티 상태 저장을 참조하세요).이 뒤에는 항상 onStart()가 따라옵니다.	아니요	onStart()
onRestart()	액티비티가 중단되었다가 다시 시작되기 직전에 호출됩니다.이 뒤에는 항상 onStart()가 따라옵니다.	아니요	onStart()
onStart()	액티비티가 사용자에게 표시되기 직전에 호출됩니다.액티비티가 전경으로 나오면 onResume()이 뒤에 따라오고, 액티비티가 숨겨지면 onStop()이 뒤에 따라옵니다.	아니요	onResume() 또는 onStop()
onResume()	액티비티가 시작되고 사용자와 상호작용하기 직전에 호출됩니다. 이 시점에서 액티비티는 액티비티 스택의 맨 위에 있으며, 사용자가 정보를 입력하고 있습니다.이 뒤에는 항상 onPause()가 따라옵니다.	아니요	onPause()
onPause()	시스템이 다른 액티비티를 재개하기 직전에 호출됩니다. 이 메서드는 일반적으로 데이터를 유지하기 위해 저장되지 않은 변경 사항을 커밋하는 데, 애니메이션을 비롯하여 CPU를 소모하는 기타 작업을 중단하는 등 여러 가지 용도에 사용됩니다. 이 메서드는 무슨 일을 하든 매우 빨리 끝내야 합니다. 이것이 반환될 때까지 다음 액티비티가 재개되지 않기 때문입니다.액티비티가 다시 전경으로 돌아오면 onResume()이 뒤에 따라오고 액티비티가 사용자에게 보이지 않게 되면 onStop()이 뒤에 옵니다.	예	onResume() 또는 onStop()
onStop()	액티비티가 더 이상 사용자에게 표시되지 않게 되면 호출됩니다. 이것은 액티비티가 소멸되고 있기 때문에 일어날 수도 있고, 다른 액티비티 (기존 것이든 새로운 것이든)가 재개되어 이것을 덮고 있기 때문일 수도 있습니다.액티비티가 다시 사용자와 상호작용하면 onRestart()가 뒤에 따라오고 액티비티가 사라지면 onDestroy()가 뒤에 따라옵니다.	예	onRestart() 또는 onDestroy()
onDestroy()	액티비티가 소멸되기 전에 호출됩니다. 이것이 액티비티가 받는 마지막 호출입니다. 이것이 호출될 수 있는 경우는 액티비티가 완료되는 중이기 때문(누군가가 여기에 finish()를 호출해서)일 수도 있고, 시스템이 공간을 절약하기 위해 액티비티의 이 인스턴스를 일시적으로 소멸시키는 중이기 때문일 수도 있습니다. 이와 같은 두 가지 시나리오는 isFinishing() 메서드로 구분할 수 있습니다.	예	없음

• 출처 https://developer.android.com/guide/components/activities.html

Life Cycle 확인



Activity 상태 저장 1

- Activity 상태의 임시 저장 → 앱 실행 중
  - Activity가 화면 회전 등을 할 때 유지해야 할 정보
  - onSaveInstanceState() 에 상태 보관
  - onCreate() 또는 onRestoreInstanceState() 복구 수행
- Activity 상태 영구 저장 → 앱 종료 전
  - 앱 종료 후에도 유지되어야 할 정보
  - onPause()에서 Preference, DB, 파일 등에 상태 보관
  - onCreate()에서 복구 수행
- onSaveInstanceState() vs. onPause()
  - onPause() 는 생명주기 관련 메소드이며 호출이 보장되므로 영구적인 데이터 보관 시 사용
  - onSaveInstanceState() 는 앱 실행 중에만 유지할 데이터 보관 시 사용

Activity 상태 저장 2

앱 실행 중 data 보관

상태 저장 구현 위치 – onSaveInstanceState()

```
override fun onSaveInstanceState(outState: Bundle) {
    super.onSaveInstanceState(outState)
    outState.putInt("x", binding.myView.x)
    outState.putInt("y", binding.myView.y)
}
```

- 인수로 전달받은 Bundle 에 상태 정보 저장

상태 복구 구현 위치

- onCreate()

```
override fun onCreate(savedInstanceState: Bundle?) {
    ...
    binding.myView.x = savedInstanceState?.getInt("x") ?: 200
```

- 저장한 Bundle이 없을 경우 → null → 초기값 200 지정

- onRestoreInstanceState()

```
override fun onRestoreInstanceState(savedInstanceState: Bundle) {
    super.onRestoreInstanceState(savedInstanceState)
    binding.myView.x = savedInstanceState.getInt("x")
    binding.myView.y = savedInstanceState.getInt("y")
}
```

- Bundle 에 저장한 값 복원

Activity 상태 저장 3

상태 정보 영구 저장

- Preference 사용
  - 앱 설정 정보 등을 저장하기 위한 용도로 사용
  - XML 형식으로 휴대폰 내부 저장소의 특정위치에 저장
- 저장 구현 위치 – onPause()

```
override fun onPause() {
    super.onPause()
    val pref : SharedPreferences = getSharedPreferences("save_state", 0)
    val editor : SharedPreferences.Editor = pref.edit()
    editor.putInt("y", binding.myView.y)
    editor.commit()
}
```

- save_state 에 값 저장
- Editor 를 사용하여 저장

- 복구 구현 위치 – onCreate()

```
override fun onCreate(savedInstanceState: Bundle?) {
    ...
    val pref = getSharedPreferences("save_state", 0)
    binding.myView.y = pref.getInt("y", 200)
```

참고 – Log 클래스 사용

Log

- 원하는 정보 (변수값, 실행 상태 등)을 지정하여 출력 → LogCat 창에서 확인

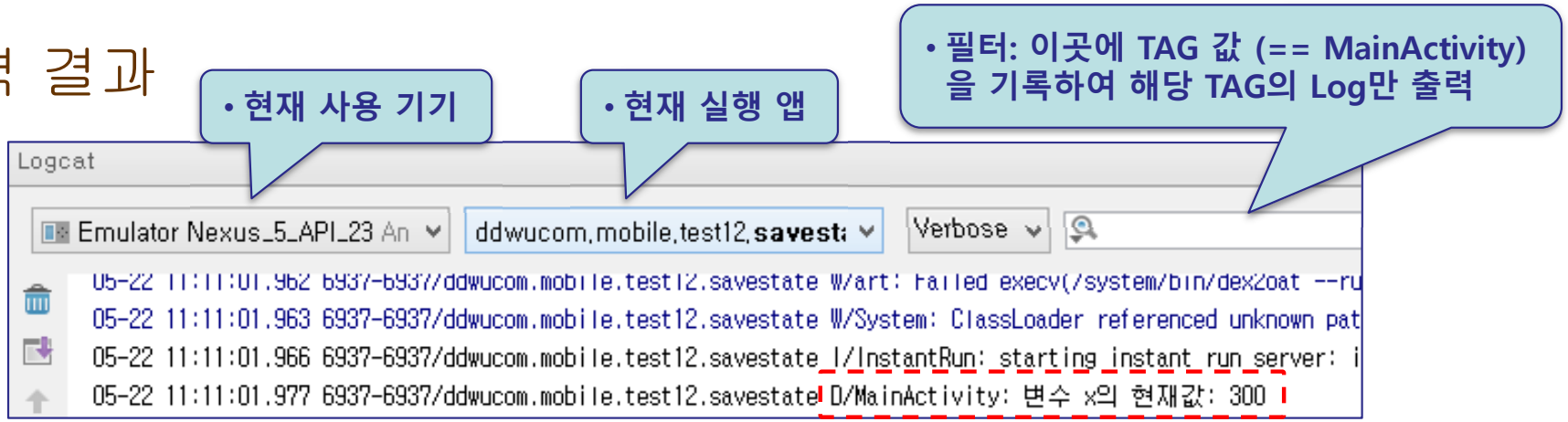
```
Log.d ( String: tag, String: message [, Throwable tr] )
// d: debug, e: error, w: warning, i: information, v: verbose
```

- 예:

```
// log 를 구분하기 위한 Tag 값은 보통 액티비티 명으로 지정
val TAG = "MainActivity";
...
Log.d (TAG, "변수 x 의 현재 값: $x");
```

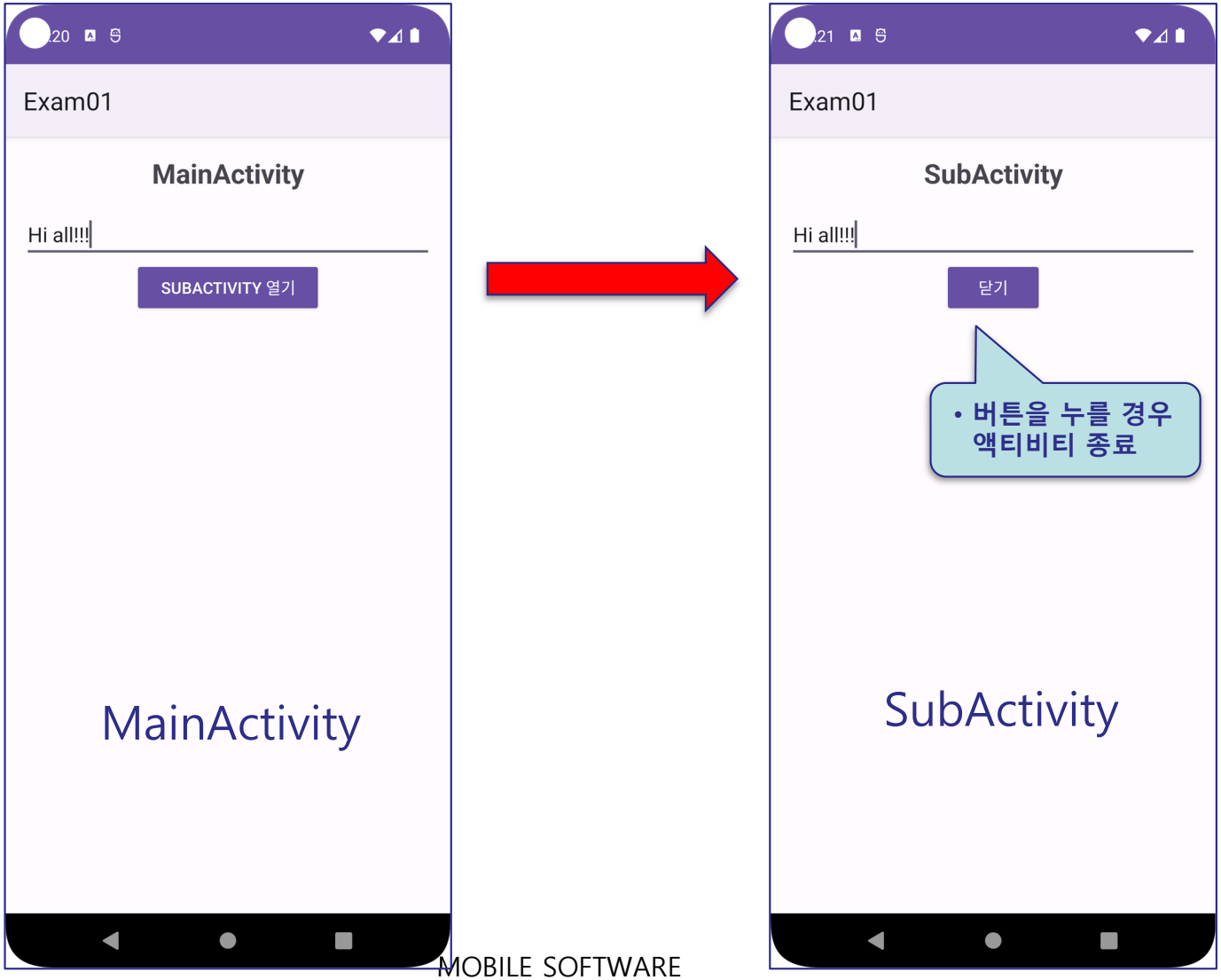
찍을 String 값

- 출력 결과



실습 1

- 다음과 같이 MainActivity 에서 문자열 입력 후 버튼을 누르면 SubActivity를 호출하고, MainActivity에서 입력한 문자열을 출력하도록 앱을 구현하시오. (Exam01 사용)



실습 2

- 추가 버튼을 누를 경우 음식을 추가하는 액티비티 호출 후 새로운 음식을 입력하여 목록에 출력하는 앱을 구현하시오. (FoodExam_init 사용)
  - AddFoodActivity가 새 FoodDto 반환