안드로이드 프로그래밍

2025년 2학기

Android 앱의 화면 흐름 설계 (Building User Screen Flows)

강의 목표

- Android Activity Lifecycle 이해
- 로그로 콜백 순서 추적 실습
- Activity 상태 저장·복원 이해 및 구현

액티비티(Activity)란 무엇인가?

- 사용자 인터페이스(UI)를 담는 단일 화면을 나타내는 구성 요소
- 사용자가 앱에서 보는 모든 화면은 기본적으로 하나의 액티비티 또는 여러 개의 프래그먼트를 포함한 액티비티
- 앱 실행 시 가장 먼저 호출되는 컴포넌트
  - AndroidManifest.xml에서 <intent-filter>로 지정
- Activity는 AppCompatActivity를 상속받아 UI를 관리

안드로이드 시스템의 역할

- 시스템은 Activity의 생명주기를 직접 제어
- 안드로이드 OS는 메모리 부족 또는 구성 변경(Configuration Changes)과 같은 상황에서 앱의 안정성과 사용자 경험을 관리하기 위해 앱의 액티비티를 생성, 중단, 재시작 또는 소멸(kill)시키는 역할
- 개발자는 시스템이 호출하는 생명주기 콜백(Lifecycle Callbacks)에 응답하여 앱 상태를 관리해야
  - onCreate → onStart → onResume → onPause → onStop → onDestroy
- 콜백(Callback) 메커니즘: OS가 특정 상태 변화가 발생했음을 앱에게 알리기 위해 액티비티 클래스의 사전에 정의된 메서드를 호출하는 방식

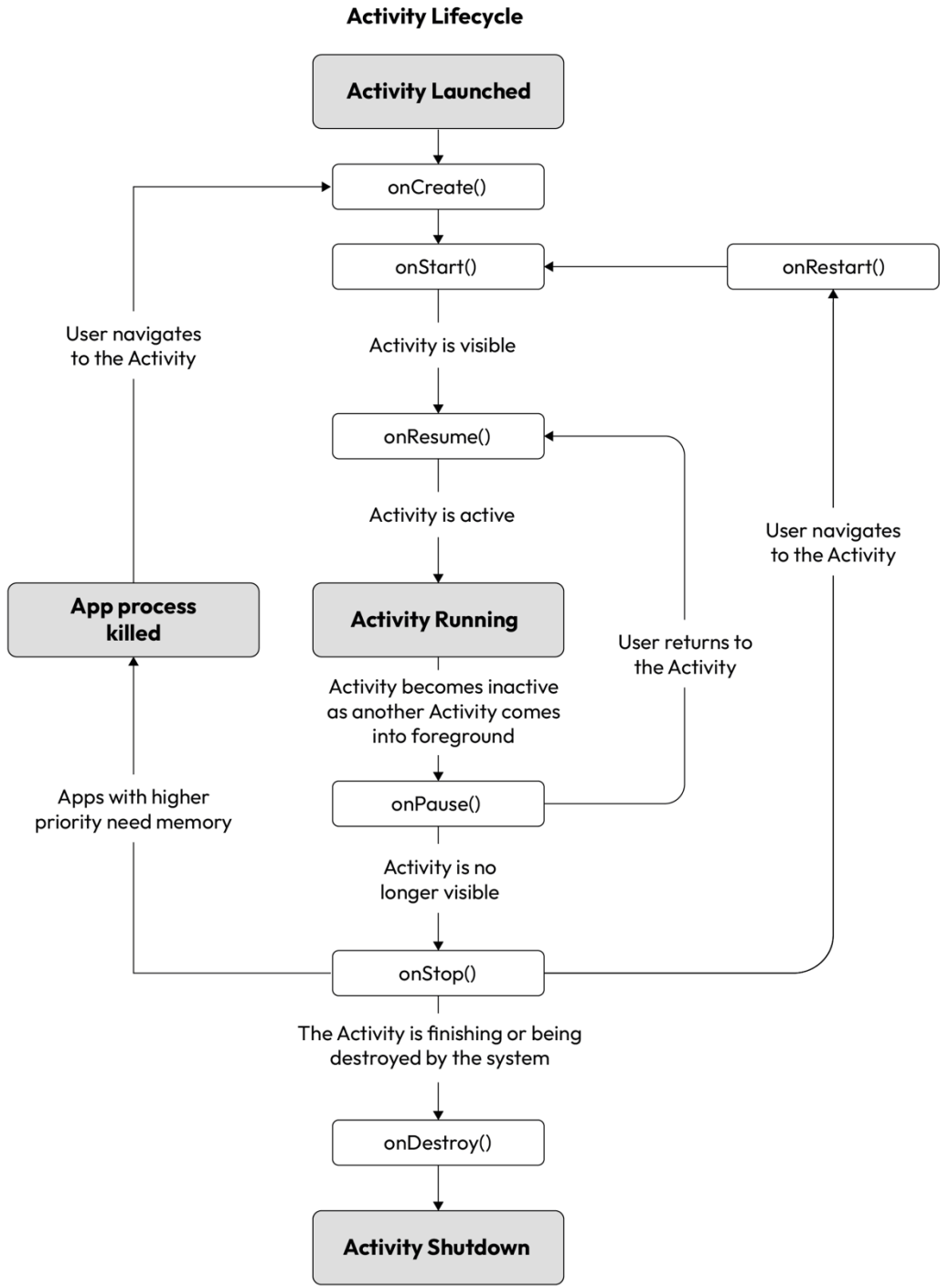
액티비티 생명주기의 개념

- 정의: 액티비티가 실행되는 동안 거치는 일련의 상태 변화 과정
- 초기화부터 화면 표시 준비, 완전히 표시됨, 숨겨짐, 백그라운드 전환, 그리고 소멸까지의 단계
- 콜백(Callback) 함수
  - 각 단계마다 액티비티의 상위 클래스(parent)에서 호출되는 특정 함수
  - 개발자는 이 함수들을 오버라이드(override)하여 해당 상태 변화에 대응하는 코드를 작성

주요 생명주기 상태 및 콜백 (개요)

- (1) 생성 (Created)
- (2) 시작 (Started)
- (3) 재개/활성화 (Resumed/Active)
- (4) 일시 중지 (Paused)
- (5) 중단 (Stopped)
- (6) 소멸 (Destroyed)

- 이 상태 변화에 대응하여 onCreate(), onStart(), onResume(), onPause(), onStop(), onDestroy() 콜백이 호출됨



Activity Lifecycle 상세 (1/4)

- onCreate()
  - 호출 시점: 액티비티가 처음 생성될 때 단 한 번 호출됨
  - 역할
    - 액티비티의 레이아웃을 설정 (setContentView()), 필요한 초기화 작업 수행
      - UI가 사용자에게 표시되기 전에 준비하는 단계
    - savedInstanceState: Bundle?
      - 액티비티가 이전에 소멸되었을 때(예: 화면 회전) 저장된 상태 데이터를 담고 있으며,
      - 이를 이용해 상태를 복원할 수 있음
      - 처음 실행 시에는 이 값이 null

Activity Lifecycle 상세 (2/4)

- onStart()
  - 액티비티가 사용자에게 보이게 될 때 호출됨
  - onCreate() 완료 후, 또는 onRestart() 이후에 호출됨
- onResume()
  - 액티비티 생성의 마지막 단계
  - 백그라운드에서 포그라운드로 다시 돌아올 때 호출됨
  - 이 콜백이 완료되면 액티비티는 활성(Active) 상태가 되어 사용자 이벤트를 받을 준비가 됨

Activity Lifecycle 상세 (3/4)

- onPause()
  - 액티비티가 백그라운드로 전환되기 시작하거나 다른 다이얼로그/액티비티가 포그라운드에 나타날 때 호출됨
  - 이 시점부터는 사용자 상호작용을 중단
  - onPause()는 매우 빠르게 실행되어야 함. 그렇지 않으면 다음 액티비티의 시작이 지연되어 사용자 경험을 해침
- onStop()
  - 액티비티가 완전히 숨겨졌을 때 호출됨
  - 백그라운드 전환 또는 다른 액티비티가 완전히 덮었을 때
  - 불필요한 리소스 해제, 네트워크 요청 중단

Activity Lifecycle 상세 (4/4)

- onRestart()
  - 액티비티가 중단되었다가(Stopped) 다시 시작될 때 (onStart() 직전에) 호출됨
  - 재시작 vs. 재 생성
    - 홈 버튼을 눌러 백그라운드에 갔다가 돌아올 때 재시작되며 onRestart()가 호출됩니다. 화면 회전 시에는 재 생성되므로 onRestart()는 호출되지 않습니다.
- onDestroy()
  - 액티비티가 소멸되기 직전에 호출됨
  - finish()가 명시적으로 호출되거나 시스템 자원이 부족할 때 발생

onStart()와 onRestart() 콜백이 분리된 이유는?

1. 최적화된 리소스 관리
   - onStart()는 액티비티가 처음 생성되었거나(onCreate() 다음) 또는 완전히 중단된 상태(onStop())에서 재활성화될 때 호출됨
   - onRestart()는 액티비티가 onStop() 상태였다가 다시 사용자에게 보이게 될 때 onStart() 직전에 호출됨
   - 시스템은 액티비티가 완전히 새로 만들어지는 상황(onCreate -> onStart)과 잠시 숨겨졌다가 돌아오는 상황(onRestart -> onStart)을 구분하여, 메모리 및 프로세스 관리 작업을 최적화
2. 상태 이력(History) 구분
   - onStart()는 액티비티가 "보이는" 상태로 진입하는 일반적인 단계
   - onRestart()는 액티비티가 "이전에 한 번 중단(Stopped)되었다"는 이력을 가지고 다시 시작됨을 나타냄
     - 이 이력 정보는 개발자가 코드를 통해 이전 세션에서 중단했던 특정 작업을 재개할 때 유용

onRestart()와 onStart()의 역할 대비

- 화면에 보이기 시작하는 데 필요한 리소스 초기화나 데이터 로딩은 onStart() 또는 onResume()에서 처리하는 것이 가장 보편적이며 권장되는 방법
- onRestart()는 특수한 재시작 상황에 대한 예외적인 로직 처리가 필요할 때만 사용해야

구분	onRestart()	onStart()
호출 시점	onStop() 이후 액티비티가 다시 포그라운드로 돌아오기 시작할 때 (항상 onStart()보다 먼저 호출됨)	onCreate() 직후 또는 onRestart() 직후, 액티비티가 사용자에게 보이게 될 때
발생 시나리오	1. 홈 버튼 누름 (앱 백그라운드 전환) 후 다시 앱으로 돌아올 때. 2. 다른 액티비티로 이동했다가 뒤로 가기 버튼으로 돌아올 때.	1. 앱이 처음 실행될 때. 2. onRestart()가 호출된 직후.
주요 역할	중단(Stopped) 시점에만 필요한 로직을 처리하여, 불필요한 초기화 로직을 건너뛸지 여부를 결정하는 분기점 역할.	화면에 보이기 시작하는 모든 경우에 공통으로 필요한 로직 처리. (예: UI 업데이트 준비, 애니메이션 시작 준비)

onRestart() 콜백에서 할 수 있는 일

1. 리소스 조건부 재로드 (Re-loading Resources)
   - onStop() 상태일 때 해제했던 리소스(예: 대용량 이미지, 특정 네트워크 리스너)를 onRestart()에서 로드하여 재사용 효율을 높일 수
2. 화면 표시 상태 확인 및 분기 처리
   - 사용자가 앱을 나갔다가 돌아왔을 때, 이전에 보이지 않던 상태와 구분하여 특별한 처리
   - 예시: 일정 시간 경과 여부를 확인하여 앱 잠금(Lock) 화면을 다시 띄우는 로직
3. 데이터 무효화 방지 또는 업데이트
   - 앱이 백그라운드에 있는 동안 외부에서 데이터가 변경되었을 가능성이 있다면, onRestart()에서 해당 데이터를 다시 로드할지 결정

데이터 지속성 및 상태 관리 (1/4)

- 상태 저장의 필요성: 구성 변경(Configuration Change)문제
  - 화면 회전(orientation)과 같은 구성 변경 시 액티비티는 기본적으로 소멸 후 재 생성됨
- 이론적 근거: 다양한 기기 및 화면 모드에 최적화된 리소스(예: 가로/세로 레이아웃)를 자동으로 로드하기 위해서
- 그러나 이는 임시 데이터를 잃는 결과를 초래

데이터 지속성 및 상태 관리 (2/4)

- 상태 저장 콜백: onSaveInstanceState(outState: Bundle)
- 호출 시점: 시스템에 의해 강제 종료될 가능성이 있을 때 호출됨
- Bundle 사용: Bundle은 Parcelable 인터페이스를 구현하며, OS가 앱 프로세스 외부의 메모리(System Space)에 저장하기에 적합한 형태로 데이터를 직렬화(Serialize)하여 보존
  - 코드 예시: outState.putString("KEY", "VALUE")

데이터 지속성 및 상태 관리 (3/4)

- 상태 복원 방법 (2가지)
  - 방법 1: onCreate()의 savedInstanceState: Bundle? 매개변수 사용 (가장 흔함)
  - 방법 2: onRestoreInstanceState(savedInstanceState: Bundle) 콜백 사용 (onStart() 이후 호출됨)

데이터 지속성 및 상태 관리 (4/4)

- 구성 변경 처리 옵션
- (1) 상태 저장/복원 (권장)
- (2) ViewModel 사용 (최신 권장)
- (3) AndroidManifest 설정 변경 (비권장)

- [심화] android:configChanges: 매니페스트에 이 속성을 설정하면 시스템의 재 생성 동작을 개발자가 직접 제어할 수 있으나, 시스템의 기본 리소스 관리 기능을 상실하므로 신중하게 사용해야