

안드로이드 시험 대비 노트

모바일 프로그래밍 · 서술형 대비 요약 / 예상 문제 / 치트시트

[참고] 표시는 공지된 범위 밖이지만 같은 단원이라 함께 정리한 내용입니다. 직접 정리한 노트이며 원본 강의자료가 아닙니다.

시험 범위 · 문제 유형

10월 10일 수정 공지 기준 · 모두 주관식 서술형

자료	시험 범위	노트
수업자료 1	p.56 프로젝트 구조 설명부터 ~ p.62	1
수업자료 2	p.9~11 액티비티 라이프사이클, 기능/실습 2(p.27~43), 기능/실습 3(p.44~57)	2 · 3 · 4
수업자료 3	기능/실습 4(p.3~21), 기능/실습 5는 레이아웃까지(p.22~32)	5 · 6

범위에서 빠지는 부분

- 1장 안드로이드 소개, 설치, 에뮬레이터 생성, 설정 (수업자료 1 p.1~55)
- 2장 액티비티/리소스 (태스크·액티비티 스택 p.4~6, 리소스 p.14~26) — **액티비티 생명주기는 포함**
- 3장 기능/실습 5의 데이터 전달(리턴 처리) 기능 코드 (ActivityResultLauncher.registerForActivityResult.setResult, p.33~39)

문제 유형

- 모든 문제는 주관식 서술형
- 클래스, 메소드, 레이아웃, 부분 코드 설명, 프로그램을 수행하는 데 알아야 하는 내용 중심
- 전체 코드 설명 / 코드 작성 문제는 출제하지 않음

생명주기 범위 — 공지대로 p.9~11만 시험 범위로 본다. 같은 단원의 p.7(개요)·p.8(3가지 상태)·p.12~13(상태 저장)은 노트와 문제에 **[참고]**로 표시했다.

Part 1. 요약 노트

1. 프로젝트 구조 · Manifest · 레이아웃 기본

수업자료 1 · p.56~62

프로젝트 폴더 구조

폴더	역할
manifests/AndroidManifest.xml	애플리케이션 수행에 필요한 필수 정보
java	소스코드 (<code>androidTest</code> 는 단위 테스트용)
res/drawable	그려질 수 있는 객체 (이미지)
res/layout	화면 디자인(레이아웃) 저장
res/mipmap	앱 아이콘 배치
res/values	문자열, 배열, 색상 등 (themes: 스타일 정보)
Gradle Scripts	빌드 스크립트

AndroidManifest.xml

- 애플리케이션의 **이름, 버전 등의 정보**를 저장
- 애플리케이션에서 선언한 **구성 요소·기능·요구사항**을 기록
— 앱에 적재된 **모든 컴포넌트**를 기술하는 파일
- 외부 라이브러리에 대한 정의 포함

AndroidManifest.xml (핵심 부분)

xml

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="My Application"
    android:theme="@style/Theme.MyApplication">
    <activity
        android:name=".MainActivity"
        android:exported="true"
        android:windowSoftInputMode="adjustResize">
        <intent-filter>
            <action
                android:name="android.intent.action.MAIN" />
            <category
                android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
```

속성/태그

의미

allowBackup	백업 및 복원 설정
icon , label	사용할 아이콘, 타이틀(앱 이름) 라벨
activity android:name	액티비티 클래스 이름 정의. 정의(등록)되지 않은 액티비티는 실행 불가
exported	외부 앱이나 시스템이 이 액티비티를 실행할 수 있도록 허용하는 설정
windowSoftIn putMode	키보드에 대한 화면 반응 정의 (adjustResize)
intent- filter	이 액티비티를 동작시키는 인텐트를 결정. MAIN + LAUNCHER = 메인(시작) 액티비티임을 선언

레이아웃 XML (res/layout/....xml)

- 사용자 인터페이스(UI)를 작성하는 파일. 보기 모드: Code / Split / Design
- `layout_width` , `layout_height` : View의 너비·높이.
`match_parent` = 부모 객체를 기준으로, `wrap_content` = 필요한 만큼
- `android:text` : TextView가 가지는 메시지
- `xmlns:android="..."` : 안드로이드 속성을 사용한다고 정의한 네임스페이스
- 제공되는 레이아웃이 여러 종류 — Linear, Frame, Relative
...

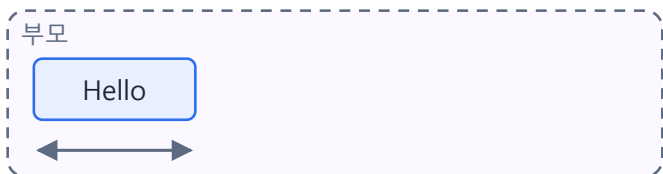
match_parent

부모 너비만큼



wrap_content

글자만큼



0dp (match_constraint)

양쪽 제약 사이를 채움



layout_width 값에 따라 같은 위젯이 차지하는 가로 크기. `0dp` 는 `ConstraintLayout`에서 양쪽 제약 사이(여기서는 parent와 margin 16dp)를 꽉 채운다는 뜻으로, 실습 2의 입력란에 쓰였다.

기본 MainActivity 코드

MainActivity.java

java

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable(this);
        setContentView(R.layout.activity_main);

        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.
            main),
                (View v, WindowInsetsCompat insets) -> {
                    Insets systemBars =
insets.getInsets(WindowInsetsCompat.Type.systemBars());
                    v.setPadding(systemBars.left,
systemBars.top,
                                systemBars.right,
systemBars.bottom);
                    return insets;
                });
    }
}
```

**extends
AppCompatActivity**

기본 액티비티 생성. 여러
액티비티 존재 가능.
AppCompatActivity를
상속해서 사용

onCreate()

어플리케이션(액티비티) 실행.
호출되면서 액티비티가 생성됨
(→ 라이프사이클)

super.onCreate(...)

상위 클래스의 onCreate()를
호출해 정의된 기본 작업 수행

`EdgeToEdge.enable(this)`

화면 가장자리까지 콘텐츠를 표시하도록 설정

`setContentView(R.layout.activity_main)`

액티비티의 **화면을 구성**.
화면에 표시할 뷰를 설정하고,
리소스(R.layout....)를 이용

`setOnApplyWindowInsetsListener`

시스템 바(상태바·
내비게이션바) 영역을 감지해
View의 패딩을 자동 설정



앱 아이콘 터치

→ MAIN + LAUNCHER 액티비티가 첫 화면

manifests/

AndroidManifest.xml

```
<activity
  android:name=
    ".MainActivity">
  <intent-filter>
    MAIN + LAUNCHER
  </intent-filter>
```

클래스 등록

java/패키지/

MainActivity.java

```
class MainActivity
  extends AppCompatActivity {
  onCreate(...) {
    setContentView(
      R.layout.activity_main);
    btn = findViewById(
      R.id.button01);
  }
```

화면 연결 ↓

id로 찾기 ↓

res/layout/

activity_main.xml

```
<ConstraintLayout ...>
  <Button
    android:id=
      "@+id/button01"
    android:text="Clear"/>
</ConstraintLayout>
```

세 파일이 연결되는 방식. Manifest가 액티비티 클래스를 등록하고, 자바 코드는 `R.layout` 으로 화면 XML 전체를, `R.id` 로 그 안의 위젯 하나를 찾아 연결한다. 앱 아이콘을 누르면 MAIN + LAUNCHER로 선언된 액티비티가 먼저 실행된다.

시험 포인트 — “Manifest의 역할”, “res 하위 폴더 구분”, “setContentView가 하는 일”.

2. 액티비티 생명주기 (Lifecycle)

수업자료 2 · p.9~11 (p.7~8, 12~13은 참고)

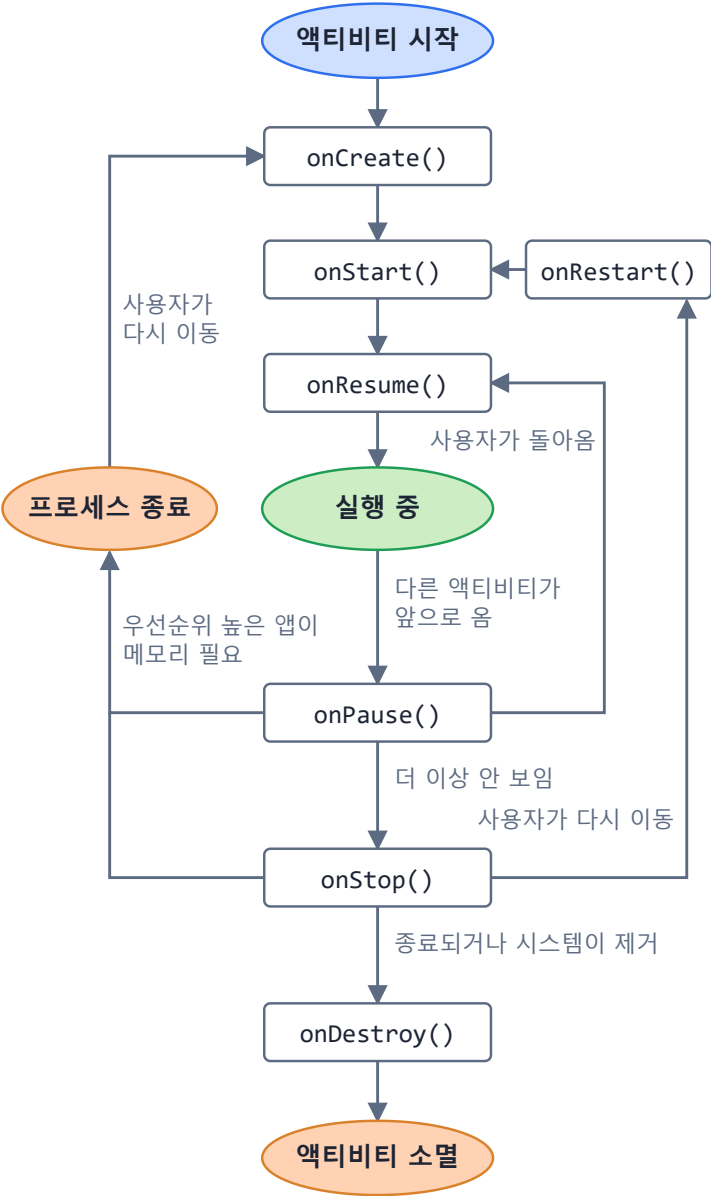
왜 생명주기가 있나 (p.7 참고)

- 모바일은 데스크탑보다 **메모리 관리가 더 효율적**이어야 함
→ 액티비티도 **생성·소멸 과정(생명주기)**이 있음
- 액티비티가 중지/정지 상태면 프로세스가 메모리에서 제거될 수 있고, 다시 보여질 때 이전 상태로 복구
- 액티비티는 **액티비티 스택**으로 관리되며, 사용자와 상호작용하는 단위(보통 전체 화면)

액티비티의 3가지 상태 (p.8 참고)

상태	특징
활성(active)/실행(running)	전경 화면. 사용자 동작에 대한 초점을 가짐 . 상호작용 가능
중지(paused)	초점은 없지만 사용자에게 일부 보임 . 상호작용 불가. 메모리가 극도로 부족하면 강제 종료
정지(stopped)	사용자에게 전혀 안 보임 . 상태·멤버 정보는 유지. 다른 컴포넌트가 메모리를 요청하면 강제 종료

생명주기 흐름도



가운데 세로줄이 기본 경로(생성 → 실행 → 소멸)이고, 오른쪽 두 화살표는 다시 돌아오는 경로, 왼쪽은 메모리 부족으로 프로세스가 종료됐다가 돌아오는 경로다. 웹에서는 아래 버튼을 눌러 상황별로 어떤 메소드가 불리는지 직접 확인할 수 있다.

콜백 메소드 정리 (p.10~11)

메소드	호출 시점 / 하는 일
<code>onCreate()</code>	액티비티가 생성될 때 처음 호출. 전역 상태의 모든 리소스 초기화(layout, data binding 등). 위젯 배치 레이아웃을 구성하고 사용자와 상호작용하는 코드를 포함
<code>onStart()</code>	초기화를 마치고 사용자에게 보여줄 준비가 되었을 때
<code>onResume()</code>	액티비티가 사용자에게 보여지고 입력을 처리할 수 있음 . 액티비티 스택의 최상위
<code>onPause()</code>	사용자와의 상호작용 중지 . 액티비티 종료 등에 대비해 상태를 저장하는 작업 수행
<code>onStop()</code>	액티비티가 더 이상 사용자에게 보이지 않음 . 더 이상 스택 최상위가 아님
<code>onDestroy()</code>	존재하는 모든 리소스 해제 . 시스템 내에 액티비티가 존재하지 않게 됨
<code>onRestart()</code>	<code>onStop()</code> 이후 정지 상태에 있던 액티비티를 다시 화면에 표시할 때 호출

상황별 호출 순서 (외워두기)

상황	호출 순서
앱 처음 실행 (Activity launched)	onCreate → onStart → onResume
홈 버튼으로 나감 (액티비티가 더 이상 안 보임)	onPause → onStop
홈에서 다시 돌아옴 (사용자가 액티비티로 다시 이동)	onRestart → onStart → onResume (onCreate는 호출 안 됨)
뒤로 가기 / finish() (액티비티 종료)	onPause → onStop → onDestroy

상태 저장하기 (p.12~13 참고)

- 정지된 액티비티는 다시 쓰일 것에 대비해 메모리에 상주하지만, **메모리가 부족하면 강제 종료됨**
- 다시 호출 시 — 강제 종료된 경우: **다시 생성한 뒤 실행 / 종료되지 않은 경우: 다시 만들 필요 없이 다시 시작해 표시**
- `onSaveInstanceState(Bundle)` : 메모리에서 제거되기 전 상태를 **Bundle**에 저장
- `onRestoreInstanceState()` : `onCreate()` / `onStart()` 이후 저장된 상태를 복원
- 이 둘은 **생명주기 메소드가 아니라서 항상 호출되지 않음** — 개발자가 재정의(override)해서 구현

시험 포인트 — ① 6~7개 콜백의 **시점+역할**을 한 줄씩 ② **onPause vs onStop** (일부 보임/전혀 안 보임) ③ **상황별 호출 순서** ④ **onRestart**는 **onStop** 이후에만, 이때 **onCreate**는 다시 안 불림.

3. 기능/실습 2 — 레이아웃 구성 기초 · 클릭 이벤트

수업자료 2 · p.27~43

무엇을 만드나

- **목적:** 레이아웃 설계의 기초, 레이아웃 구성 방법, 구성된 레이아웃에 대한 기초 코드 처리 방법, ConstraintLayout 소개
- **결과:** 텍스트 입력란에 단어 입력 가능 → **CLEAR(Clear)** 버튼을 누르면 입력한 단어가 지워지고 → **FINISH(Close)** 버튼을 누르면 액티비티 종료

ConstraintLayout

- **Design Tool** 중심의 작업. **일정한 비율 구성, 쉽고 빠른 배치** 등의 장점
- **기준점 설정** 등 세부 기능 습득이 일부 필요
- 각 레이아웃마다 장·단점이 있음: RelativeLayout, LinearLayout, TableLayout, FrameLayout 등 다양
- 다른 레이아웃으로 바꾸려면 해당 레이아웃 코드를 변경 (예: ConstraintLayout → LinearLayout)하고, 선택한 레이아웃에 따라 **추가 속성 정의**가 필요

레이아웃 작성 요점

- 텍스트 입력 위젯: **Plain Text**(= EditText) 선택. 클릭 처리 위젯: **Button** (클릭 처리를 버튼만 지원하는 건 아님)
- **Blueprint** : 위젯의 종류와 테두리를 표시 → 위젯 간 관계 설정에 유용
- Design에서 위젯을 선택해 속성을 설정, **Go To XML** 로 해당 위젯 코드로 이동 (같은 위젯이 여러 개일 때 편리)

- Plain Text 너비를 최대로: `match_constraint` = `0dp` /
Button 폭·높이: `wrap_content`
- 각 객체에 주어진 **id**로 화면의 위젯을 구분

```
<EditText
    android:id="@+id/editText01"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginStart="16dp"
    android:layout_marginTop="16dp"
    android:layout_marginEnd="16dp"
    android:hint="text type"
    android:inputType="text"
    android:minHeight="48dp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<Button
    android:id="@+id/button01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="16dp"
    android:layout_marginTop="16dp"
    android:text="Clear"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/editText01" />

<Button
    android:id="@+id/button02"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="16dp"
    android:layout_marginTop="16dp"
    android:text="Close"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/button01" />
```

@+id/이름

위젯에 id를 새로 정의.
코드에서 R.id.이름 으로 접근

`layout_constraintTop_toBottomOf`

이 위젯의 **위쪽**을 지정 위젯의 **아래쪽**에 붙임 (위치 관계 설정)

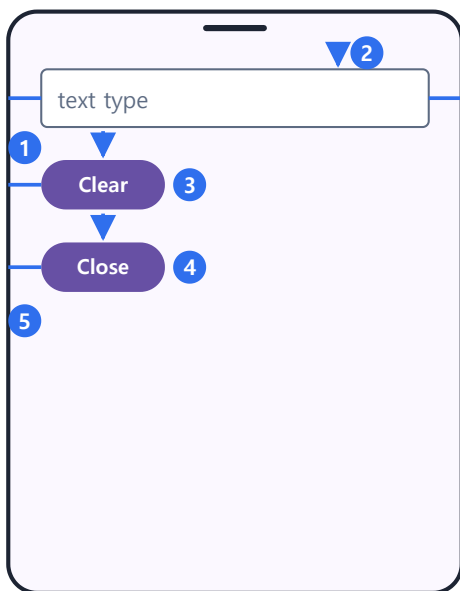
`layout_constraintStart_toStartOf="parent"`

이 위젯의 시작(왼쪽)을 부모의 시작에 맞춤

`hint / inputType`

입력 안내 문구 / 입력 종류 (text 등)

실습 2 실행 화면



- 1 editText01 Start·End → parent
양옆에 붙이고 width 0dp → 가로로 꽉 참
- 2 editText01 Top_toTopOf="parent"
화면 위쪽에 붙임 (margin 16dp)
- 3 button01 Top_toBottomOf="@+id/editText01"
입력란 바로 아래 — Clear
- 4 button02 Top_toBottomOf="@+id/button01"
Clear 버튼 아래 — Close
- 5 button01·02 Start_toStartOf="parent"
둘 다 왼쪽 정렬, 크기는 wrap_content

XML의 제약 속성이 화면 배치로 바뀌는 모습. 번호마다 왼쪽 화면의 파란 선 하나가 오른쪽 속성 하나에 대응한다.

클릭 이벤트 처리 코드

MainActivity.java (실습 2)

java

```
public class MainActivity extends AppCompatActivity
    implements View.OnClickListener {

    EditText editText01;
    Button btn01, btn02;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable(this);
        setContentView(R.layout.activity_main);
        // ... insets 처리 생략 ...

        editText01 = (EditText)
findViewById(R.id.editText01);
        btn01 = (Button) findViewById(R.id.button01);
        btn02 = (Button) findViewById(R.id.button02);

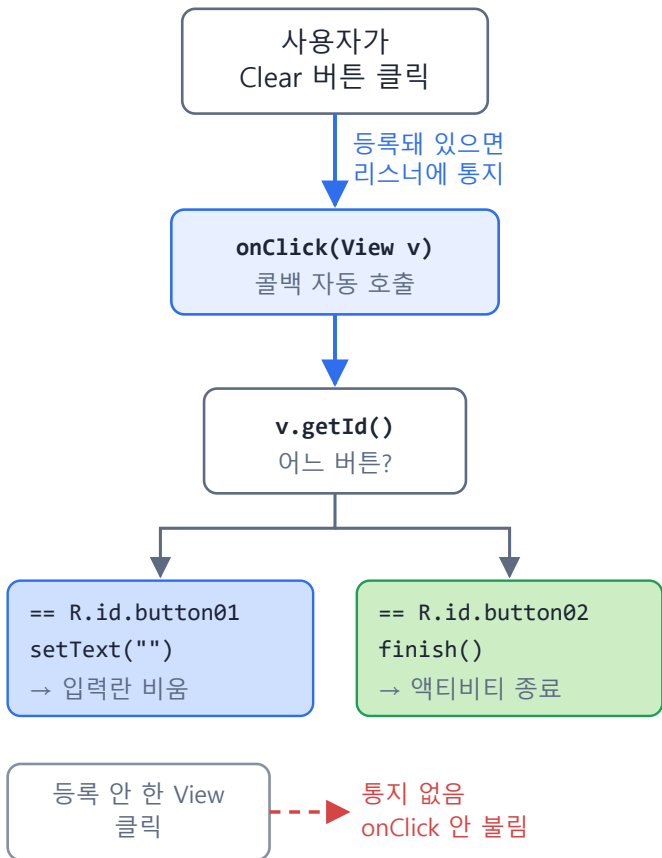
        btn01.setOnClickListener(this);
        btn02.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        if (v.getId() == R.id.button01) {
            editText01.setText("");
        }
        if (v.getId() == R.id.button02) {
            finish();
        }
    }
}
```

import

필요한 클래스 import —
자동으로 import 됨

<code>implements View.OnClickListener</code>	클릭 이벤트 처리를 위한 리스너 정의
<code>EditText editText01; Button btn01, btn02;</code>	레이아웃 객체와 연결할 변수 선언
<code>findViewById(R.id...)</code>	레이아웃 객체와 연결 (XML의 id로 객체를 찾아 변수에 대입)
<code>setOnClickListener(this)</code>	클릭 리스너에 등록 . 등록된 구성요소에 대해서만 클릭 발생 시 이벤트가 리스너에 통지됨
<code>onClick(View v)</code>	등록된 버튼이 클릭되었을 때 호출되는 콜백 메소드 . 이벤트 발생 시 처리할 내용을 작성
<code>v.getId() == R.id.button01</code>	어느 버튼이 눌렸는지 구분 (여러 대상이 같은 리스너에 등록된 경우)
<code>setText("")</code>	텍스트 설정 (빈 문자열 → 입력 내용 지움)
<code>finish()</code>	액티비티 종료



`setOnClickListener(this)` 로 등록된 버튼만 클릭이 리스너에 통지되어 `onClick()` 이 불린다. 두 버튼이 리스너 하나를 같이 쓰므로 `getId()` 로 갈라서 처리한다.

시험 포인트 — 클릭 이벤트 처리 4단계: ①

`implements View.OnClickListener` ② 변수 선언 후 `findViewById` 로 연결 ③

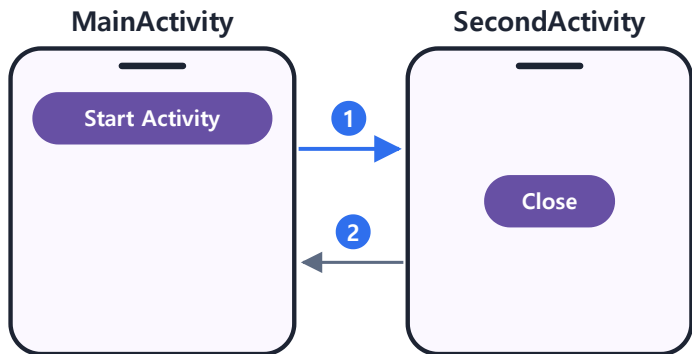
`setOnClickListener(this)` 등록 ④ `onClick()` 안에서 `getId()` 로 구분해 처리.

4. 기능/실습 3 — 액티비티 추가 · 인텐트로 실행

수업자료 2 · p.44~57

목적과 핵심

- 인텐트(Intent)를 이용한 액티비티 실행 과정 이해 —
액티비티 추가 작성 + 인텐트로 실행하는 방법
- 현재 액티비티에서 다른 액티비티 화면을 열기:
`startActivity(Intent);`
- 결과: 버튼을 클릭하면 새로운 액티비티가 실행됨

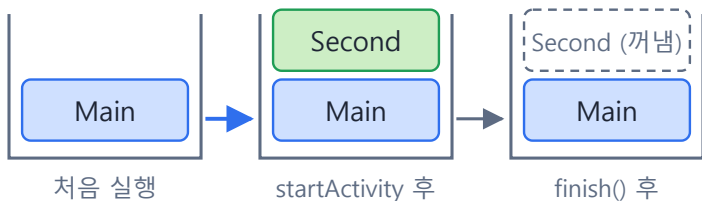


1 `new Intent(현재, 대상.class)`
 → `startActivity(intent)`로 실행

2 `finish()` → 이전 화면(Main)으로

※ Manifest에 등록된 액티비티만 실행 가능

액티비티 스택



실행된 SecondActivity가 액티비티 스택의 최상위에 오고(p.10 `onResume`), 가려진 Main은 더 이상 최상위가 아니다(p.11 `onStop`). `finish()` 로 Second를 종료하면 Main이 다시 보인다. 위쪽 두 화면은 실습 3의 실제 레이아웃 배치(가로로 짝 찬 Start Activity 버튼, 가운데 Close 버튼)를 따랐다.

진행 단계

1 **액티비티 준비** — MainActivity, SecondActivity (화면 전환을 위한 액티비티 필요)

- 2 레이아웃 준비 — activity_main.xml, activity_second.xml
(액티비티별로 레이아웃 작성)
- 3 레이아웃 작성 — Main: 새 액티비티 실행 버튼
(button01 , width 0dp + 좌우·위 parent 제약 → 위쪽에
가로로 짝 찬 "Start Activity") / Second: 종료 버튼만
(button02 "Close", 상하좌우 모두 parent 제약 → 화면
가운데)
- 4 액티비티 코드 작성 — Intent 생성 + startActivity,
종료는 finish()

액티비티 추가 방법

- 코드(패키지)에서 우클릭 → **New** → **Activity** → **Empty Views Activity** 선택 (Java Class 선택 X)
- Activity Name = SecondActivity , Layout Name = activity_second (이름은 자유)
- 결과: 액티비티(Main/Second) + 레이아웃
(activity_main.xml / activity_second.xml) 생성,
AndroidManifest.xml에 자동 등록

Java Class로 만들면 안 되는 이유 — 액티비티 마법사(Activity)로 만들어야 레이아웃 파일 생성과 Manifest 등록이 자동으로 된다. 액티비티를 호출하려면 매니페스트에 등록되어 있어야 하며, 정상적으로 추가했다면 자동으로 추가되어 있다.

```
<activity
    android:name=".SecondActivity"
    android:exported="false"
    android:windowSoftInputMode="adjustResize" />
```

exported="false"

외부 앱이나 시스템이 이
액티비티를 실행하는 것을
허용하지 않는 설정 (exported
설명은 수업자료 1 p.59)



코드 모양은 안드로이드
스튜디오 버전에 따라 차이가
있음 (자료에도 명시)

MainActivity.java — 새 액티비티 실행

java

```
Button btn01;
// onCreate 안에서
btn01 = (Button) findViewById(R.id.button01);
btn01.setOnClickListener(this);

@Override
public void onClick(View v) {
    Intent intent01 = new Intent(MainActivity.this,
    SecondActivity.class);
    startActivity(intent01);
}
```

new
Intent(MainActivity
.this,
SecondActivity.clas
s)

현재 액티비티(Context)에서
실행할 대상 클래스를 지정한
인텐트 생성

startActivity(inten
t01)

인텐트를 이용해 다른
액티비티를 실행

```
public class SecondActivity extends AppCompatActivity
    implements View.OnClickListener {

    Button btn01;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable(this);
        setContentView(R.layout.activity_second);    // 이 액
티비티의 레이아웃
        // ... insets 처리 생략 ...

        btn01 = (Button) findViewById(R.id.button02);
        btn01.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        finish();
    }
}
```

**setContentView(R.la
yout.activity_secon
d)**

**SecondActivity용 레이아웃을
연결** (Main은 activity_main)

onClick → finish()

리스너에 등록된 대상이
1개라서 대상 구분(getId)이
필요 없음. finish()로 종료 →
이전 액티비티(Main)로 돌아감

시험 포인트 — 새 화면을 띄우는 전 과정(액티비티 생성 → 레이아웃 → Manifest 등록 → Intent + startActivity)과 **Manifest에 없으면 실행 불가**. finish() 는 현재 액티비티를 끝내 이전 화면으로 복귀.

5. 기능/실습 4 — 액티비티 간 데이터 전달 (putExtra)

수업자료 3 · p.3~21

목적

- 다른 액티비티로 데이터를 전달하는 과정 습득 — Intent 객체 활용, 데이터 전달·구분 과정 확인
- 액티비티를 실행하면서 지정된 데이터를 함께 전달
(startActivity + putExtra)

MainActivity

Intent

Start SecondActivity

Close

① putExtra(키, 값)

② startActivity()



Intent intent01

대상: SecondActivity.class

키	값
"key01"	"value\nData"

③ getIntent()

④ getStringExtra(키)



SecondActivity

\n은 줄바꿈이라
두 줄로 보임

전달된 값 출력

value
Data

Button

값은 Intent 안에 (키, 값) 쌍으로 실려 가고, 받는 쪽은 `getIntent()` 로 자기를 실행한 Intent를 꺼낸 뒤 **같은 키**로 값을 찾는다. `\n` 은 줄바꿈이라 회색 TextView에 두 줄로 보인다.

레이아웃 구성

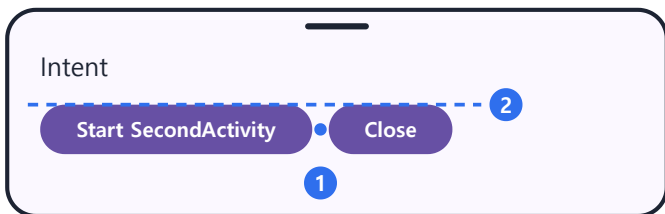
파일	구성
activity_main.xml	TextView 1개(단순 표기용, 글씨 24sp) + Button 2개 (액티비티 실행 / 종료). ConstraintLayout
button01 / button02	button01(Start SecondActivity)은 TextView 아래 (Top_toBottomOf="@+id/textView01"). button02(Close)는 Start_toEndOf="@+id/button01" + Top_toTopOf="@+id/button01" → button01 오른쪽 옆, 같은 높이에 나란히 (p.14 강조, p.12 화면)
activity_second.xml	TextView 2개(첫째: 단순 표기용 / 둘째 result_textView : 받은 데이터 표시용, 색상·크기 조절) + Button 1개(button03 , 종료)
result_textView	width·height 모두 0dp + 위는 첫 TextView 아래 (Top_toBottomOf), 아래는 버튼 위(Bottom_toTopOf)에 제약 → 그 사이 공간을 꽉 채움. background="#FFBDBDBD" , textSize="24sp"

layout_constraint A _to B Of="X"

→ 이 위젯의 A쪽을 X의 B쪽에 붙인다

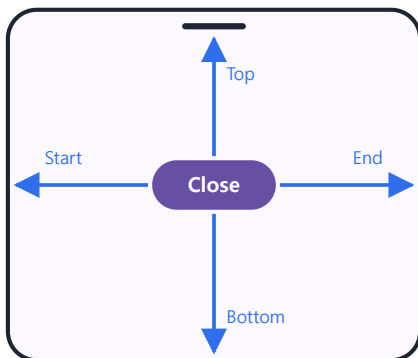
X = parent(부모) 또는 다른 위젯의 id

실습 4 · activity_main (p.12·14)



- 1 button02 Start_toEndOf="@+id/button01"
Close 왼쪽을 Start 버튼 오른쪽에 붙임 → 바로 옆
- 2 button02 Top_toTopOf="@+id/button01"
Close 위쪽을 Start 버튼 위쪽에 맞춤 → 같은 높이

실습 3 · activity_second (p.54)



- 3 button02 Top·Bottom·Start·End → "parent"
네 방향 모두 부모에 붙음 → 가로·세로 가운데

제약 속성 이름은 "이 위젯의 쪽_to상대의 쪽Of"로 읽는다. ①②는 실습 4 activity_main의 Close 버튼(p.14에서 빨간 상자로 강조), ③은 실습 3 activity_second의 Close 버튼(p.54)이다.

- 글씨 크기: `android:textSize="24sp"` — 단위 필수, 글씨 크기 단위는 **sp**
- 배경색: `android:background="#BDBDBD"` (RGB 예: `#FFBDBDBD` 또는 `#BDBDBD`)

보내는 쪽: MainActivity

MainActivity.java

java

```
btn01 = (Button) findViewById(R.id.button01);
btn02 = (Button) findViewById(R.id.button02);
btn01.setOnClickListener(this);
btn02.setOnClickListener(this);

@Override
public void onClick(View v) {
    if (v.getId() == R.id.button01) {
        Intent intent01 = new Intent(MainActivity.this,
SecondActivity.class);
        intent01.putExtra("key01", "value\nData");
        startActivity(intent01);
    }
    if (v.getId() == R.id.button02) {
        finish();
    }
}
```

**findViewById /
setOnClickListener**

버튼의 클릭 이벤트 처리를
위해 레이아웃과 연결 →
리스너에 등록

**intent01.putExtra("
key01",
"value\nData")**

Intent에 전달할 데이터를 기록.
(이름(키), 값) 쌍으로 구성.
특정 데이터를 추가해 전달할
때 사용

\n

줄바꿈 문자 — 받는 쪽
TextView에 "value" 다음 줄에
"Data"로 표시됨

받는 쪽: **SecondActivity**

SecondActivity.java

java

```
TextView result_text;
Button btn01;

// onCreate 안에서
Intent data_receive;
data_receive = getIntent();
String temp01 = data_receive.getStringExtra("key01");

result_text = (TextView) findViewById(R.id.result_textView);
result_text.setText(temp01);

btn01 = (Button) findViewById(R.id.button03);
btn01.setOnClickListener(this);

@Override
public void onClick(View v) {
    finish();
}
```

getIntent()

이 액티비티를 실행시킨
Intent를 가져옴

**getStringExtra("key
01")**

넘어온 데이터를 **네임(키)**
값으로 찾아 확인 (String 값
꺼내기)

**result_text.setText
(temp01)**

받은 값을 하단의 **TextView**에
출력

시험 포인트 — 전달 과정 3줄 요약: 보내기

`putExtra(키, 값)` → 받기 `getIntent()` → 꺼내기 `getStringExtra(키)`. 여러 데이터를 보낼 때는 **키 이름으로 구분한다**. 키가 서로 달라야 구분됨.

6. 기능/실습 5 — LinearLayout (레이아웃까지)

수업자료 3 · p.22~32

범위 안내 — 실습 5는 레이아웃까지가 시험 범위라 이 앱에서는 **LinearLayout**과 레이아웃 구성만 다룬다. 결과 리턴 기능(p.33~39)은 범위 밖.

레이아웃 목적 (p.23)

- 레이아웃 부분은 **LinearLayout**으로 구성 — LinearLayout에 대한 기초 사용법 습득

준비 · 레이아웃 구성 (p.26~28, 32)

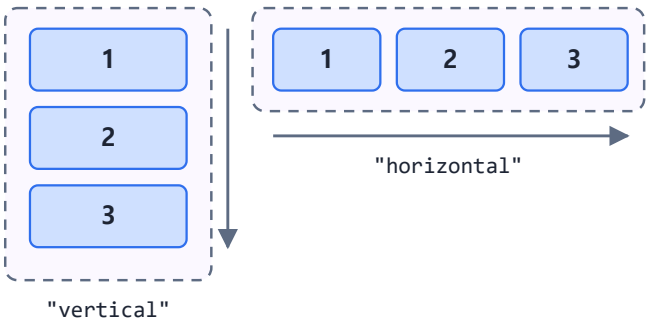
- 액티비티: MainActivity.java, SecondActivity.java / 레이아웃: activity_main.xml, activity_second.xml
- activity_main: **EditText**(보낼 값 입력) + **Button 2개**(start / close) + **TextView 2개**("Result :" 제목, 종료된 액티비티로부터 전달받은 결과를 출력하는 회색 영역 #FFBDBDBD)
- activity_second: 세로 LinearLayout 안에 **EditText**(전달받은 데이터 표기 및 편집) + **Button**(button03 "close", width match_parent)

LinearLayout (p.29~31)

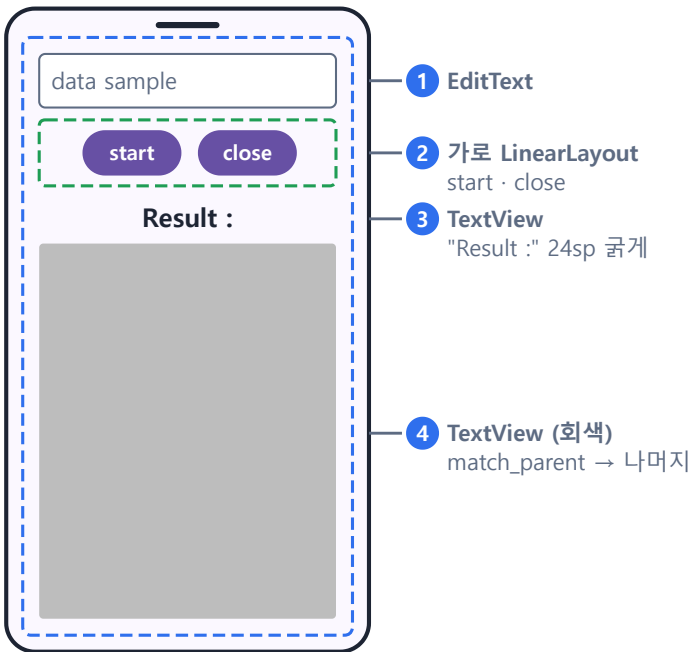
- 순서대로 한 방향으로 나열해 주는 레이아웃

- `android:orientation` : 배치 방향 결정 — `"vertical"` (세로) / `"horizontal"` (가로)
- 배치 방향을 바꾸려면 `orientation` 값을 변경
- 알아야 할 기본 속성: `android:orientation` , `android:weightSum` , `android:layout_weight` ...
- 자료의 예제에서 `weightSum` 은 **LinearLayout** 설명을 위해 적어 둔 것일 뿐 실제로는 사용하지 않음

orientation



실습 5 · activity_main



바깥 = 세로 LinearLayout (padding 8dp)

LinearLayout은 orientation 방향으로 자식을 차례대로 쌓는다.
실습 5의 activity_main은 세로 LinearLayout 안에 가로
LinearLayout(버튼 2개, `gravity="center"`)을 넣은 중첩 구조다.

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:weightSum="1"
    android:padding="8dp">

    <EditText
        android:id="@+id/editText01"
        android:inputType="text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:orientation="horizontal">
        <Button android:id="@+id/button01"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginEnd="8dp"
            android:text="start" />
        <Button android:id="@+id/button02"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="close" />
    </LinearLayout>

    <TextView
        android:id="@+id/textView01"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Result :"
        android:textAlignment="center"
        android:textSize="24sp"
        android:textStyle="bold" />

    <TextView
```

```

        android:id="@+id/textView02"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="#FFBDBDBD"
        android:textSize="18sp" />
    </LinearLayout>

```

orientation="vertical"

바깥 레이아웃: 자식들을 **세로 (위→아래)**로 나열

orientation="horizontal"

안쪽 레이아웃: 버튼 2개를 **가로로** 나열 — 자료의 "배치 방향의 변경이 필요한 경우" 예

weightSum="1"

LinearLayout 설명을 위해 적어 둔 것으로 이 예제에서는 사용하지 않음

**textView02
match_parent**

마지막 TextView가 **남은 아래 공간을 채움** → 돌려받은 결과가 표시되는 회색 영역

비교	ConstraintLayout	LinearLayout
배치 방식	위젯 간 위치 관계(제약)로 배치	한 방향으로 순서대로 나열
핵심 속성	layout_constraint...	orientation , layout_weight

Part 2. 서술형 예상 문제와 모범답안

Q1 · 프로젝트 구조

안드로이드 프로젝트의 주요 폴더/파일 구성과 각각의 역할을 서술하시오.

- manifests: AndroidManifest.xml(필수 정보)
- java: 소스코드 (androidTest는 단위 테스트)
- res: 리소스 폴더 (drawable·layout·mipmap·values)
- Gradle Scripts: 빌드 스크립트

프로젝트는 manifests, java, res, Gradle Scripts로 구성된다. manifests에는 앱 수행에 필요한 필수 정보를 담은 AndroidManifest.xml이 있고, java에는 소스코드(androidTest는 단위 테스트용)가 있다. res는 리소스 폴더로 drawable(이미지), layout(화면 디자인), mipmap(앱 아이콘), values(문자열·배열·색상, themes는 스타일)로 나뉜다. Gradle Scripts에는 빌드 스크립트가 있다.

Q2 · 프로젝트 구조

AndroidManifest.xml의 역할을 설명하고, 액티비티를 새로 추가했을 때 이 파일에 반드시 등록되어 있어야 하는 이유를 쓰시오.

- 앱의 이름·버전 등 정보 저장
- 구성 요소·기능·요구사항 선언 / 모든 컴포넌트 기술
- 정의되지 않은 액티비티는 실행 불가
- 액티비티 추가 마법사를 쓰면 자동 등록

AndroidManifest.xml은 애플리케이션의 이름, 버전 등의 정보를 저장하고, 앱에서 선언한 구성 요소·기능·요구사항과 앱에 적재된 모든 컴포넌트를 기술하는 파일이다. 액티비티의 android:name에 클래스가 정의되지 않으면 그 액티비티는 실행할 수 없으므로, 액티비티를 호출하려면 반드시 매니페스트에 등록되어 있어야 한다. Android Studio의 액티비티 추가 과정을 정상적으로 진행하면 자동으로 등록된다.

Q3 · 프로젝트 구조

Manifest의 activity 안 <intent-filter>에서 MAIN, LAUNCHER가 의미하는 바와 exported 속성의 역할을 쓰시오.

- intent-filter: 액티비티를 동작시키는 인텐트 결정
- MAIN + LAUNCHER = 메인(시작) 액티비티 선언
- exported: 외부 앱/시스템이 액티비티 실행 가능하도록 허용

intent-filter는 해당 액티비티를 동작시키는 인텐트를 결정하는 설정이며, action MAIN과 category LAUNCHER를 지정하면 그 액티비티가 앱의 메인(시작) 액티비티임을 선언하는 것이다. exported는 외부 앱이나 시스템이 이 액티비티를 실행할 수 있도록 허용할지를 정하는 설정이다.

Q4 · 프로젝트 구조

res 폴더 하위의 drawable, layout, mipmap, values의 차이를 설명하시오.

- drawable: 그려질 수 있는 객체(이미지)
- layout: 화면 디자인/구성 XML
- mipmap: 앱 아이콘
- values: 문자열·배열·색상 등 (themes는 스타일)

drawable은 화면에 그려질 수 있는 객체인 이미지를 저장하고, layout은 화면 디자인(구성 형태)을 정의한 XML을 저장한다. mipmap은 앱 아이콘이 배치되는 폴더이며, values는 문자열·배열·색상 등의 값을 저장하고 themes에는 스타일 정보가 들어간다.

Q5 · 프로젝트 구조

레이아웃 XML에서 layout_width/layout_height에 쓰이는 match_parent와 wrap_content의 차이를 설명하시오.

- match_parent: 부모 객체를 기준으로(부모 크기에 맞춤)
- wrap_content: 내용에 필요한 만큼만

match_parent는 부모 객체를 기준으로 부모의 크기만큼 채우도록 지정하는 값이고, wrap_content는 View가 내용을 표시하는 데 필요한 만큼의 크기만 차지하도록 지정하는 값이다.

Q6 · 프로젝트 구조

기본 MainActivity의 onCreate()에서 super.onCreate()와 setContentView(R.layout.activity_main)이 각각 하는 일을 설명하시오.

- onCreate: 액티비티가 생성될 때 호출
- super.onCreate: 상위 클래스의 정의된 작업 수행
- setContentView: 화면에 표시할 뷰(레이아웃) 설정
- R.layout.activity_main = 리소스 이용

onCreate()는 액티비티가 생성되면서 호출되는 메소드다. super.onCreate()는 상위 클래스의 onCreate()를 호출해 정의된 기본 작업을 수행하게 하고, setContentView(R.layout.activity_main)은 액티비티의 화면을 구성하는 역할로 화면에 표시할 뷰를 설정하며 이때 res/layout의 activity_main 리소스를 사용한다.

Q7 · 액티비티 생명주기 (Lifecycle)

[참고] 액티비티의 3가지 상태(활성/중지/정지)를 각각 설명하시오.

- 활성(running): 전경, 초점, 상호작용 가능
- 중지(paused): 초점 없음, 일부 보임, 상호작용 불가
- 정지(stopped): 전혀 안 보임, 상태 정보는 유지
- 메모리 부족 시 시스템이 강제 종료할 수 있음

활성(실행) 상태는 전경 화면에서 사용자 동작에 대한 초점을 가지며 상호작용이 가능한 상태다. 중지 상태는 초점은 없지만 사용자에게 일부 보이는 상태로 상호작용은 불가하며 메모리가 극도로 부족하면 강제 종료된다. 정지 상태는 사용자에게 전혀 보이지 않지만 모든 상태와 멤버 정보는 유지하고, 다른 컴포넌트가 메모리를 요청하면 시스템에 의해 강제 종료될 수 있다.

Q8 · 액티비티 생명주기 (Lifecycle)

액티비티 생명주기 콜백 메소드 onCreate()가 호출되는 시점과 하는 일을 설명하시오.

- 액티비티가 생성될 때 처음으로 호출
- 전역 상태의 모든 리소스 초기화 (예: layout, data binding)
- 위젯이 배치된 레이아웃을 구성하고, 위젯이 사용자와 상호작용하는 코드를 포함

onCreate()는 액티비티가 생성될 때 처음으로 호출되는 메소드다. 전역 상태의 모든 리소스를 초기화하는데, 예를 들어 layout과 data binding 등이 있다. 액티비티는 버튼, 리스트, 체크박스, 입력 표시줄 같은 위젯들이 배치된 레이아웃을 구성하고, 위젯들이 사용자와 상호작용하는 코드를 포함한다.

Q9 · 액티비티 생명주기 (Lifecycle)

액티비티 생명주기 콜백 메소드 onStart()가 호출되는 시점을 설명하시오.

- 액티비티가 초기화 과정을 마친 후
- 사용자에게 보여줄 준비가 되었을 때 호출

onStart()는 액티비티가 초기화 과정을 마친 후 사용자에게 보여줄 준비가 되었을 때 호출되는 메소드다.

Q10 · 액티비티 생명주기 (Lifecycle)

액티비티 생명주기 콜백 메소드 onResume()이 호출되었을 때 액티비티의 상태를 설명하시오.

- 액티비티가 사용자에게 보여짐
- 사용자의 입력을 처리할 수 있음
- 액티비티 스택의 최상위에 위치

onResume()이 호출되면 액티비티가 사용자에게 보여지고 사용자의 입력을 처리할 수 있는 상태가 된다. 이때 액티비티는 액티비티 스택의 최상위에 위치한다.

Q11 · 액티비티 생명주기 (Lifecycle)

액티비티 생명주기 콜백 메소드 onPause()가 호출되는 시점과 이 메소드에서 하는 작업을 설명하시오.

- 사용자와의 상호작용이 중지될 때
- 액티비티 종료 등을 대비해 상태를 저장하는 등의 작업 수행

onPause()는 사용자와의 상호작용이 중지될 때 호출되는 메소드다. 액티비티가 종료되는 경우 등을 대비해 상태를 저장하는 등의 작업을 수행한다.

Q12 · 액티비티 생명주기 (Lifecycle)

액티비티 생명주기 콜백 메소드 onStop()이 호출되었을 때 액티비티의 상태를 설명하시오.

- 더 이상 액티비티가 사용자에게 보이지 않음
- 더 이상 액티비티 스택의 최상위에 위치하지 않음

onStop()이 호출되면 액티비티는 더 이상 사용자에게 보이지 않고, 더 이상 액티비티 스택의 최상위에 위치하지 않는다.

Q13 · 액티비티 생명주기 (Lifecycle)

액티비티 생명주기 콜백 메소드 onDestroy()가 하는 일을 설명하시오.

- 존재하는 모든 리소스를 해제
- 시스템 내에 액티비티가 존재하지 않게 됨

onDestroy()는 액티비티에 존재하는 모든 리소스를 해제하는 메소드로, 호출된 후에는 시스템 내에 액티비티가 존재하지 않게 된다.

Q14 · 액티비티 생명주기 (Lifecycle)

액티비티 생명주기 콜백 메소드 onRestart()가 호출되는 시점을 설명하시오.

- onStop() 호출 이후 정지 상태에 있던 액티비티를
- 다시 화면에 표시할 때 호출

onRestart()는 onStop() 메소드가 호출된 이후 정지 상태에 있던 액티비티를 다시 화면에 표시할 때 호출되는 메소드다.

Q15 · 액티비티 생명주기 (Lifecycle)

onPause()와 onStop()의 차이를 설명하시오.

- onPause: 상호작용 중지, 아직 일부 보일 수 있음
- onStop: 사용자에게 완전히 안 보임, 스택 최상위 아님
- onPause에서 상태 저장 등 수행

onPause()는 사용자와의 상호작용이 중지되는 시점으로 액티비티가 일부는 아직 보일 수 있고, 종료 등에 대비해 상태를 저장하는 작업을 수행한다.
onStop()은 액티비티가 더 이상 사용자에게 전혀 보이지 않고 액티비티 스택의 최상위에도 위치하지 않을 때 호출된다. 즉 onPause가 먼저 불리고, 완전히 가려지면 onStop이 이어서 호출된다.

Q16 · 액티비티 생명주기 (Lifecycle)

액티비티를 처음 실행해서 뒤로 가기로 종료할 때까지의 콜백 호출 순서를 쓰고, 각 단계에서의 액티비티 상태를 설명하시오.

- onCreate → onStart → onResume (실행)
- 실행 중 = 스택 최상위·상호작용 가능
- onPause → onStop → onDestroy (종료)
- onDestroy 후 액티비티 소멸

실행 시 onCreate → onStart → onResume 순서로 호출되어 onResume 이후 액티비티가 스택 최상위에서 사용자 입력을 처리하는 실행 상태가 된다. 뒤로 가기로 종료하면 onPause → onStop → onDestroy 순서로 호출되고, onDestroy에서 모든 리소스가 해제되어 액티비티가 소멸한다.

Q17 · 액티비티 생명주기 (Lifecycle)

홈 버튼을 눌러 앱을 나갔다가 다시 돌아왔을 때 호출되는 콜백 순서를 쓰고, `onRestart()`가 필요한 이유를 설명하시오.

- 나갈 때: `onPause` → `onStop`
- 돌아올 때: `onRestart` → `onStart` → `onResume`
- `onCreate`는 다시 호출되지 않음
- `onRestart`: `onStop` 이후 정지 상태 액티비티를 다시 표시

홈 버튼으로 나가면 `onPause` → `onStop`이 호출되어 정지 상태가 되고, 다시 돌아오면 `onRestart` → `onStart` → `onResume` 순서로 호출된다. 정지 상태의 액티비티가 아직 메모리에 남아 있으면 새로 만들 필요가 없으므로 `onCreate`는 호출되지 않고, `onStop()` 이후 정지 상태에 있던 액티비티를 다시 화면에 표시할 때 `onRestart()`가 호출된다.

Q18 · 액티비티 생명주기 (Lifecycle)

`onStart()`와 `onResume()`의 차이를 설명하시오.

- `onStart`: 사용자에게 보여줄 준비가 되었을 때
- `onResume`: 실제로 보여지고 입력 처리 가능, 스택 최상위

`onStart()`는 액티비티가 초기화를 마치고 사용자에게 보여줄 준비가 되었을 때 호출되고, `onResume()`은 액티비티가 실제로 사용자에게 보여지며 사용자의 입력을 처리할 수 있는, 액티비티 스택 최상위 상태가 될 때 호출된다.

Q19 · 액티비티 생명주기 (Lifecycle)

[참고] onSaveInstanceState()와

onRestoreInstanceState()의 역할과, 이 두 메소드가 생명주기 메소드가 아니라는 말의 의미를 설명하시오.

- 메모리 부족으로 프로세스가 제거될 수 있음
- 제거 전 Bundle에 상태 저장 / 이후 복원
- 항상 호출되지 않음 → 개발자가 override해서 구현

onPause나 onStop 이후 메모리가 부족하면 프로세스가 제거될 수 있어서, onSaveInstanceState()는 제거되기 전에 Bundle 객체에 액티비티의 동적 상태를 저장하고, onRestoreInstanceState()는 onCreate/onStart 이후 저장된 상태를 복원한다. 이들은 생명주기 메소드가 아니기 때문에 항상 호출되지는 않으며, 개발자가 해당 메소드를 재정의해 구현해야 한다.

ConstraintLayout의 특징과 장점을 설명하고, 다른 레이아웃의 종류를 쓰시오.

- Design Tool 중심 작업
- 일정한 비율 구성, 쉽고 빠른 배치
- 기준점 설정 등 세부 기능 습득 필요
- Relative/Linear/Table/Frame 등 다른 레이아웃

ConstraintLayout은 Design Tool 중심으로 작업하며 일정한 비율 구성과 쉽고 빠른 배치가 가능하다는 장점이 있지만 기준점 설정 같은 세부 기능을 익혀야 한다. 각 레이아웃마다 장단점이 있으며 RelativeLayout, LinearLayout, TableLayout, FrameLayout 등 다양한 레이아웃이 있다.

Q21 · 기능/실습 2

버튼 클릭 이벤트를 처리하기 위한 코드 작성 단계를 순서대로 설명하시오. (실습 2 기준)

- implements View.OnClickListener (리스너 정의)
- 변수 선언 + findViewById로 레이아웃 객체 연결
- setOnClickListener(this)로 리스너 등록
- onClick(View v)에서 getId()로 구분해 처리

먼저 클래스에 implements View.OnClickListener로 클릭 리스너를 정의하고, 레이아웃 객체와 연결할 변수를 선언한다. onCreate에서 findViewById(R.id....)로 각 객체를 변수에 연결하고 setOnClickListener(this)로 리스너에 등록한다. 등록된 구성요소가 클릭되면 onClick(View v)가 호출되므로 그 안에서 v.getId()로 어떤 버튼인지 구분해 setText()나 finish() 같은 처리 내용을 작성한다.

Q22 · 기능/실습 2

리스너에 등록하는 이유는 무엇이며, 등록하지 않은 View를 클릭하면 어떻게 되는지 설명하시오.

- 클릭 발생 시 이벤트가 리스너에 통지됨
- 등록된 구성요소에 대해서만 통지
- 미등록 View는 onClick 호출 안 됨

setOnClickListener로 등록된 구성요소에 대해서만 클릭이 발생했을 때 이벤트가 리스너에 통지되어 onClick() 콜백이 호출된다. 따라서 등록하지 않은 View를 클릭하면 이벤트가 리스너로 전달되지 않아 onClick()이 호출되지 않는다.

Q23 · 기능/실습 2

EditText의 layout_width를 0dp(match_constraint)로 지정하는 의미와 wrap_content와의 차이를 설명하시오.

- 0dp = match_constraint (ConstraintLayout)
- 제약(양쪽 constraint) 범위만큼 최대한로 늘어남
- wrap_content = 필요한 만큼

ConstraintLayout에서 0dp는 match_constraint를 의미하며, 위젯이 양쪽 제약 사이의 가능한 공간을 최대한 채우도록 너비를 설정한다(실습에서 텍스트 입력란의 너비를 최대한로 설정). 반면 wrap_content는 내용이 필요한 만큼의 크기만 차지하며 실습의 버튼에 사용했다.

Q24 · 기능/실습 2

다음 코드에서 **button01**, **button02**를 눌렀을 때 각각 어떤 동작이 일어나는지 쓰시오.

```
public void onClick(View v) {  
    if (v.getId() == R.id.button01) {  
        editText01.setText(""); }  
    if (v.getId() == R.id.button02) { finish(); }  
}
```

- button01: setText("")로 입력란 내용 지움
- button02: finish()로 액티비티 종료
- getId()로 클릭된 버튼 구분

button01(Clear)을 누르면 editText01.setText("")가 실행되어 입력한 텍스트가 지워진다. button02(Close)를 누르면 finish()가 실행되어 현재 액티비티가 종료된다. 두 버튼이 같은 리스너를 쓰므로 v.getId()로 클릭된 버튼을 구분한다.

실습 3에서 버튼으로 SecondActivity를 열고 닫을 때 쓰이는 ① AndroidManifest.xml 등록 ② Intent 객체 ③ startActivity() ④ finish()가 각각 하는 일을 설명하시오.

- Manifest: 액티비티 등록 — 등록 안 된 액티비티는 실행 불가 (Activity 메뉴로 추가하면 자동 등록)
- Intent: 실행할 대상(현재 액티비티 → SecondActivity.class)을 담는 객체
- startActivity(): Intent로 다른 액티비티 실행
- finish(): 현재 액티비티 종료 → 이전 화면(Main)으로 복귀

AndroidManifest.xml은 앱의 구성 요소를 기술하는 파일로, 액티비티를 실행하려면 여기에 등록되어 있어야 한다(New → Activity → Empty Views Activity로 추가하면 자동 등록된다). Intent는 다른 액티비티를 실행하기 위한 객체로, new Intent(MainActivity.this, SecondActivity.class)처럼 현재 액티비티와 실행할 대상 클래스를 지정한다. startActivity()는 이 Intent를 받아 SecondActivity를 실행한다. finish()는 현재 액티비티를 종료하는 메소드로, SecondActivity에서 호출하면 이전 화면인 MainActivity로 돌아간다.

Q26 · 기능/실습 3

Intent의 역할을 설명하고, new Intent(MainActivity.this, SecondActivity.class)의 두 인자의 의미를 쓰시오.

- 인텐트를 이용해 다른 액티비티를 실행
- 첫째: 현재 액티비티(실행하는 쪽)
- 둘째: 실행할 대상 액티비티 클래스
- startActivity(intent)로 실제 실행

Intent는 다른 액티비티를 실행하기 위해 사용하는 객체로, startActivity(Intent)에 넘기면 해당 액티비티가 실행된다. new Intent(MainActivity.this, SecondActivity.class)에서 첫 번째 인자는 현재(실행을 요청하는) 액티비티이고, 두 번째 인자는 실행할 대상 액티비티의 클래스이다.

Q27 · 기능/실습 3

액티비티를 Java Class가 아닌 Activity 추가 메뉴(Empty Views Activity)로 만들어야 하는 이유와, Manifest에 등록하지 않으면 어떻게 되는지 설명하시오.

- 레이아웃 파일 생성 + Manifest 등록이 자동
- Manifest에 정의되지 않은 액티비티는 실행 불가

Activity 추가 메뉴로 만들면 액티비티 클래스와 함께 레이아웃 파일이 생성되고 AndroidManifest.xml에도 액티비티 정보가 자동으로 등록된다. 매니페스트에 등록되지 않은 액티비티는 호출(실행)할 수 없다.

Q28 · 기능/실습 3

SecondActivity에서 onClick에 대상 구분(getId() 비교)이 필요 없는 이유를 쓰시오.

- 리스너에 등록된 대상이 1개
- onClick이 호출되면 그 버튼뿐

SecondActivity에는 리스너에 등록된 버튼이 1개뿐이므로 onClick()이 호출되면 그 버튼이 눌린 것이 확실해 대상을 구분할 필요가 없고, 곧바로 finish()를 호출하면 된다.

Q29 · 기능/실습 3

실습 3의 activity_second.xml에서 Close 버튼(button02)이 화면 가운데에 놓이는 이유를 제약 속성으로 설명하시오.

```
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toTopOf="parent"
```

- 위·아래·시작·끝 네 방향 모두 parent에 제약
- 가로는 Start·End, 세로는 Top·Bottom이 모두 부모에 걸림
- 크기는 wrap_content → 부모 가운데에 배치

button02는 크기가 wrap_content이고, Top_toTopOf·Bottom_toBottomOf·Start_toStartOf·End_toEndOf가 모두 parent로 지정되어 위·아래·왼쪽·오른쪽 네 방향 모두 부모에 제약이 걸린다. 가로와 세로 양쪽이 모두 부모 기준으로 잡히므로 버튼은 화면의 가운데에 놓인다.

액티비티 간 데이터 전달(실습 4)에 쓰이는 putExtra(), getIntent(), getStringExtra() 메소드가 각각 하는 일을 설명하시오.

- putExtra(키, 값): 보내는 쪽에서 Intent에 전달할 데이터 기록, (이름(키), 값) 쌍
- getIntent(): 받는 쪽에서 자신을 실행시킨 Intent를 가져옴
- getStringExtra(키): 그 Intent에서 이름(키)으로 문자열 데이터를 찾아 꺼냄

putExtra()는 보내는 쪽(MainActivity)에서 Intent 객체에 전달할 데이터를 기록하는 메소드로, 데이터는 (이름(키), 값) 쌍으로 구성된다. 데이터를 담은 Intent를 startActivity()로 넘기면 대상 액티비티가 실행되면서 데이터도 함께 전달된다. getIntent()는 받는 쪽(SecondActivity)에서 자신을 실행시킨 Intent를 가져오는 메소드이고, getStringExtra()는 그 Intent에서 보낼 때 쓴 이름(키) 값으로 넘어온 문자열 데이터를 찾아 꺼내는 메소드이다. 꺼낸 값은 setText()로 TextView에 출력한다.

Q31 · 기능/실습 4

여러 개의 데이터를 전달할 때 각 데이터를 어떻게 구분하는지 설명하시오.

- putExtra의 이름(키)으로 구분
- 받을 때 같은 키로 getXxxExtra
- 키는 서로 달라야 함

putExtra()는 (이름(키), 값) 형태로 데이터를 저장하므로 서로 다른 키 이름을 붙여 여러 데이터를 구분한다. 받는 쪽에서는 같은 키 이름으로 getStringExtra(키) 등을 호출해 원하는 데이터를 찾는다.

다음 코드로 전달된 데이터가 **SecondActivity**의 **TextView**에 어떻게 표시되는지 쓰시오.

```
// MainActivity
intent01.putExtra("key01", "value\nData");

// SecondActivity
String temp01 = getIntent().getStringExtra("key01");
result_text.setText(temp01);
```

- key01 키로 값을 꺼냄
- \n은 줄바꿈
- 첫 줄 value, 둘째 줄 Data

key01 키로 전달된 문자열 "value\nData"를 꺼내 TextView에 설정한다. \n은 줄바꿈 문자이므로 화면에는 첫 줄에 "value", 그 아랫줄에 "Data"가 두 줄로 표시된다.

Q33 · 기능/실습 4

텍스트 크기와 배경색을 지정하는 속성과 사용하는 단위/형식을 설명하시오.

- android:textSize + 단위 필수, 글씨는 sp
- android:background (RGB 예: #BDBDBD 또는 #FFBDBDBD)

글씨 크기는 android:textSize 속성으로 지정하며 단위를 반드시 써야 하고 글씨 크기 단위는 sp를 사용한다(예: 24sp). 배경색은 android:background 속성에 RGB 색상값(예: #BDBDBD 또는 #FFBDBDBD)을 지정한다.

실습 4의 activity_main.xml에서 Close 버튼(button02)에 지정된 다음 제약 속성이 하는 일을 설명하시오.

```
app:layout_constraintStart_toEndOf="@+id/button01"  
app:layout_constraintTop_toTopOf="@+id/button01"
```

- Start_toEndOf: button02의 시작(왼쪽)을 button01의 끝(오른쪽)에 붙임
- Top_toTopOf: button02의 위쪽을 button01의 위쪽에 맞춤
- 결과: 두 버튼이 같은 높이로 가로로 나란히

layout_constraintStart_toEndOf="@+id/button01"은 button02의 시작(왼쪽)을 button01의 끝(오른쪽)에 붙여 button01 바로 오른쪽에 놓고,
layout_constraintTop_toTopOf="@+id/button01"은 button02의 위쪽을 button01의 위쪽에 맞춘다.
그래서 Close 버튼은 Start SecondActivity 버튼 옆에 같은 높이로 나란히 배치된다.

Q35 · 기능/실습 5

LinearLayout의 특징과 android:orientation 속성을 설명하시오.

- 순서대로 한 방향으로 나열
- orientation으로 배치 방향 결정
- vertical(세로) / horizontal(가로)

LinearLayout은 자식 View들을 순서대로 한 방향으로 나열해 주는 레이아웃이다. android:orientation 속성으로 배치 방향을 결정하는데, "vertical"이면 세로로, "horizontal"이면 가로로 나열한다.

Q36 · 기능/실습 5

ConstraintLayout과 LinearLayout의 차이를 비교하여 설명하시오.

- Constraint: 위젯 간 위치 관계(제약)로 배치, Design Tool 중심
- Linear: 한 방향 순서 나열 (orientation)
- 각 레이아웃의 장단점

ConstraintLayout은 위젯들 사이의 위치 관계(제약)와 기준점을 지정해 배치하며 Design Tool 중심으로 일정한 비율 구성과 빠른 배치가 쉽다. LinearLayout은 orientation(vertical/horizontal)에 따라 자식들을 한 방향으로 순서대로 나열한다. 각 레이아웃마다 장단점이 있다.

LinearLayout 구성을 위해 알아야 하는 기본 속성 3가지를 쓰고, 자료의 예제에서 **weightSum**은 어떻게 쓰였는지 설명하시오.

- android:orientation, android:weightSum, android:layout_weight
- 예제의 weightSum은 LinearLayout 설명을 위해 기술한 것, 실제 미사용

LinearLayout 구성을 위해 알아야 하는 기본 속성에는 android:orientation, android:weightSum, android:layout_weight 등이 있다. 자료의 예제에서 weightSum은 LinearLayout 설명을 위해 적어 둔 것일 뿐 실제로는 사용하지 않았다.

**실습 5의 activity_main.xml 레이아웃 구조를 설명하시오.
(어떤 레이아웃 안에 무엇이 어떤 순서로 배치되는가)**

- 바깥: 세로(vertical) LinearLayout
- EditText → 가로 LinearLayout(start·close) → TextView
"Result :" → 회색 TextView 순서
- 안쪽 LinearLayout은 horizontal
- 마지막 TextView는 match_parent로 남은 공간을 채워
결과 출력

바깥은 orientation이 vertical인 LinearLayout이라
자식들이 위에서 아래로 차례대로 놓인다. 맨 위에
보낼 값을 입력하는 EditText, 그 아래에 orientation이
horizontal인 LinearLayout을 넣어 start·close 버튼
2개를 가로로 배치하고, 이어서 "Result :" 제목
TextView와 배경색 #FFBDBDBD인 TextView를 둔다.
마지막 TextView는 높이가 match_parent라 남은 아래
공간을 채우며, 종료된 액티비티로부터 돌려받은
결과를 출력하는 영역이 된다.

Part 3. 치트시트

생명주기 한 줄 요약

메소드	한 줄
onCreate	생성될 때 처음 호출 · 리소스/레이아웃 초기화
onStart	보여줄 준비 완료
onResume	보이고 입력 처리 가능 · 스택 최상위
onPause	상호작용 중지 · 상태 저장
onStop	전혀 안 보임
onRestart	onStop 이후 다시 표시할 때
onDestroy	리소스 해제 · 소멸

상황별 호출 순서

상황	순서
실행	onCreate → onStart → onResume
홈 버튼	onPause → onStop
홈에서 복귀	onRestart → onStart → onResume
뒤로가기/finish	onPause → onStop → onDestroy

자주 나오는 코드 조각

하고 싶은 일	코드
레이아웃 연결 (화면 설정)	<code>setContentView(R.layout.activity_main);</code>
위젯 객체 찾기	<code>btn = (Button) findViewById(R.id.button01);</code>
클릭 리스너 등록	<code>btn.setOnClickListener(this);</code>
클릭 콜백	<code>public void onClick(View v) { ... }</code>
클릭된 버튼 구분	<code>if (v.getId() == R.id.button01) { ... }</code>
텍스트 설정	<code>textView.setText("...");</code>
액티비티 종료	<code>finish();</code>
다른 액티비티 실행	<code>startActivity(new Intent(MainActivity.this, SecondActivity.class));</code>
데이터 담기	<code>intent.putExtra("key01", "값");</code>
데이터 받기	<code>getIntent().getStringExtra("key01");</code>

XML 속성 모음

속성	의미
<code>android:id="@+id/이름"</code>	위젯 id 정의 → <code>R.id.이름</code>
<code>layout_width/height</code>	<code>match_parent</code> 부모 기준 / <code>wrap_content</code> 필요한 만큼 / <code>0dp</code> (<code>match_constraint</code>)
<code>android:text</code>	표시 문자열
<code>android:textSize="24sp"</code>	글씨 크기 (단위 필수, sp)
<code>android:background</code>	배경색 (#BDBDBD)
<code>android:hint</code> / <code>inputType</code>	입력 안내 문구 / 입력 종류
<code>layout_constraintint..._to...Of</code>	ConstraintLayout 위치 관계 (예: <code>Top_toBottomOf</code>)
<code>android:orientation</code>	LinearLayout 배치 방향 (<code>vertical</code> / <code>horizontal</code>)
<code>weightSum</code> / <code>layout_weight</code>	LinearLayout 구성을 위해 알아야 하는 기본 속성

Manifest 한눈에

항목	의미
<code><application></code>	앱 전체 설정 (icon, label, theme, allowBackup)
<code><activity android:name></code>	액티비티 등록 — 없으면 실행 불가
<code>exported</code>	외부 앱/시스템의 실행 허용 여부
<code>intent-filter MAIN+LAUNCHER</code>	메인(시작) 액티비티 선언
<code>windowSoftInputMode</code>	키보드에 대한 화면 반응