

01. 레이아웃 (Layout)



안드로이드 레이아웃

- ◆ Layout 개요
- ◆ LinearLayout
- ◆ RelativeLayout
- ◆ FrameLayout
- ◆ 그 밖의 레이아웃

레이아웃 관리

- ◆ 레이아웃 중첩
- ◆ 앱 화면 구현 방법
- ◆ 실행 중 레이아웃 속성 변경
- ◆ ViewBinding

Layout

 ViewGroup의 일종으로 다른 View들을 내부에 배치하는 컨테이너 역할 수행

- ◆ 위젯 또는 다른 레이아웃을 내부에 배치하여 다양한 화면 구성
- ◆ 일반적으로 화면 상에 직접 보이지 않음
- ◆ Layout 도 하나의 View 취급

 주요 Layout

- ◆ LinearLayout/RelativeLayout/FrameLayout/GridLayout
- ◆ 추가 레이아웃: ConstraintLayout



기본 & 필수

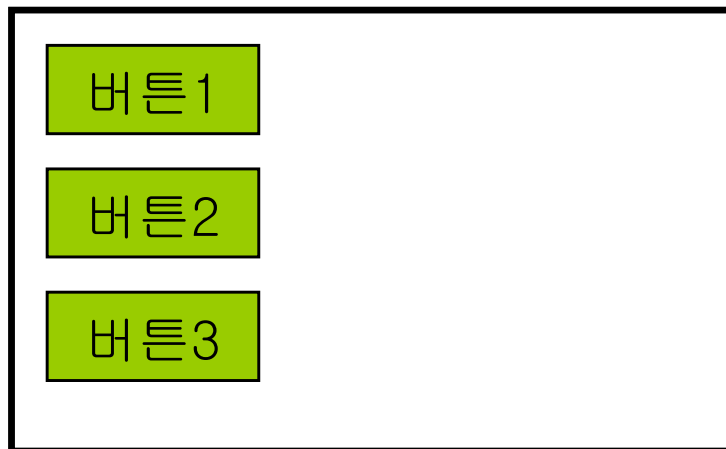
LinearLayout 1

가장 간단한 레이아웃으로 가로 또는 세로의 순서대로 항목을 배치

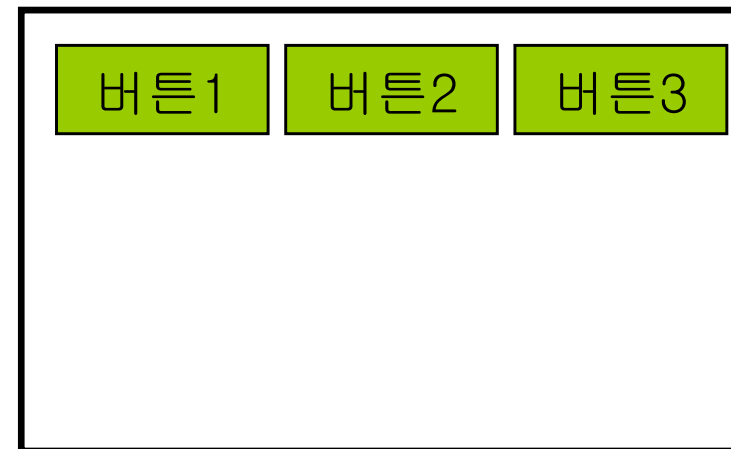
주요 속성 *Linear Layout만 가진 속성!*

orientation: vertical | horizontal

내부 View의 배치 방향 결정



vertical



horizontal

LinearLayout 2

주요 속성 (계속)

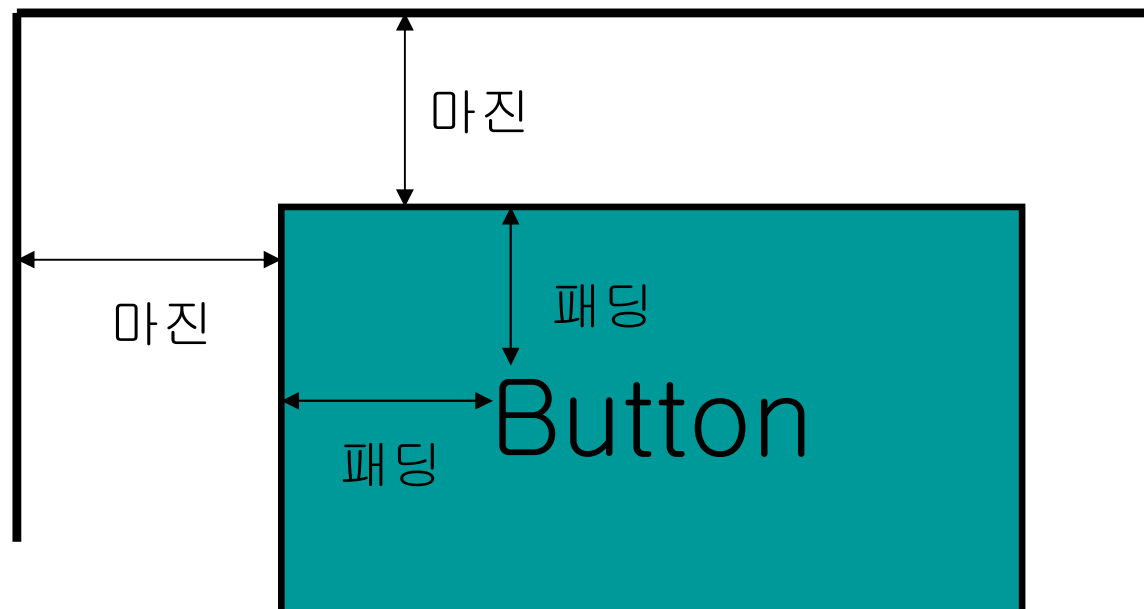
- ◆ gravity: 내부 View(또는 값)의 수직/수평 방향 배치 결정
 - 예: TextView 내부의 **글자 배치**
- ◆ layout_gravity: 레이아웃에 View 자신의 수직/수평 방향 배치 결정

```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:gravity="top"
    android:text="Button" />
```

LinearLayout 3

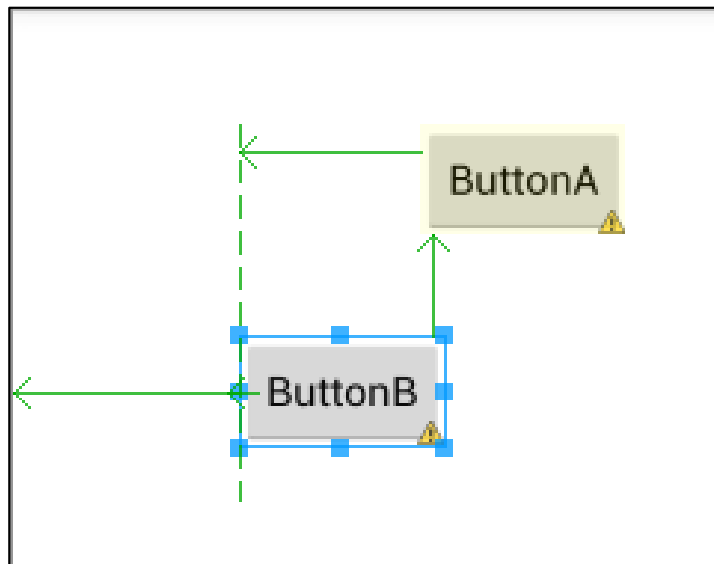
주요 속성 (계속)

- ◆ **baselineAligned**: 레이아웃에 배치한 View의 아래 부분 맞춤 활성화 여부 κ^2
- ◆ **layout_weight**: 레이아웃의 공간을 어느 정도 비중으로 차지하느냐를 결정 → 0일 경우 본래 크기, 1 이상이면 다른 뷰와의 비율에 따라 레이아웃에 배치
- ◆ **layout_margin**: 레이아웃과 뷰 사이의 간격
- ◆ **padding**: 뷰와 내부 내용물 사이의 간격



RelativeLayout

뷰와 뷰를 담고 있는 레이아웃(부모 뷰), 그리고 다른 뷰와의 상대적인 위치 관계로 배치 → **id 필수**



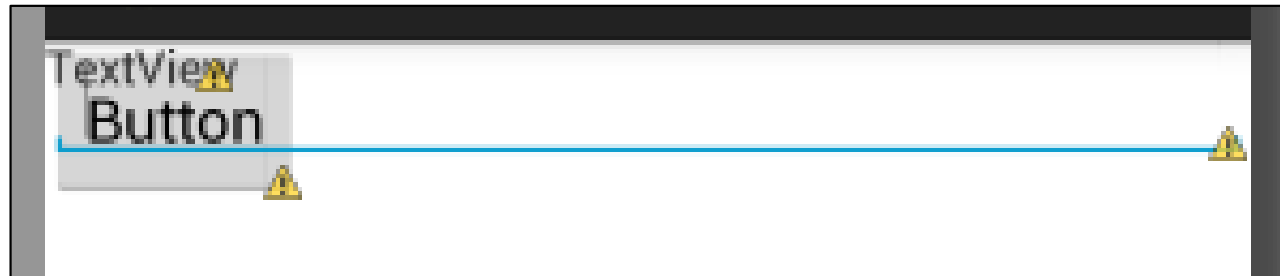
```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/button2"
    android:layout_alignParentTop="true"
    android:layout_marginLeft="78dp"
    android:layout_marginTop="50dp"
    android:text="ButtonA" />

<Button
    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_below="@+id/button1"
    android:layout_marginLeft="97dp"
    android:layout_marginTop="43dp"
    android:text="ButtonB" />
```

FrameLayout

레이아웃의 좌측 상단에 모든 뷰들을 겹쳐서 배치

- ◆ 앱 실행 중 `addView/removeView` 메소드를 사용하여 뷰들을 추가 및 삭제
- ◆ 뷰의 `visibility` 속성을 사용하여 한 화면에서 여러 화면을 번갈아 보여주고 싶을 때 사용



배치 상태

그 밖의 레이아웃

❏ AbsoluteLayout

↗ 안씀

- ◆ 배치하는 View의 좌표를 절대값으로 지정
- ◆ 안드로이드 기기 별로 다양한 해상도를 갖고 있으므로 사용에 적합하지 않음

❏ TableLayout

- ◆ 표 형식으로 View 를 내부에 배치
 - 가로: TableRow 의 개수만큼 행 생성
 - 세로: TableRow 에 포함된 View 의 개수만큼 열 생성

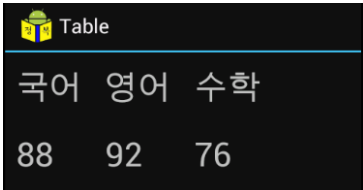


Table		
국어	영어	수학
88	92	76

발전형

❏ GridLayout

- ◆ Android API Level 14 부터 사용 가능

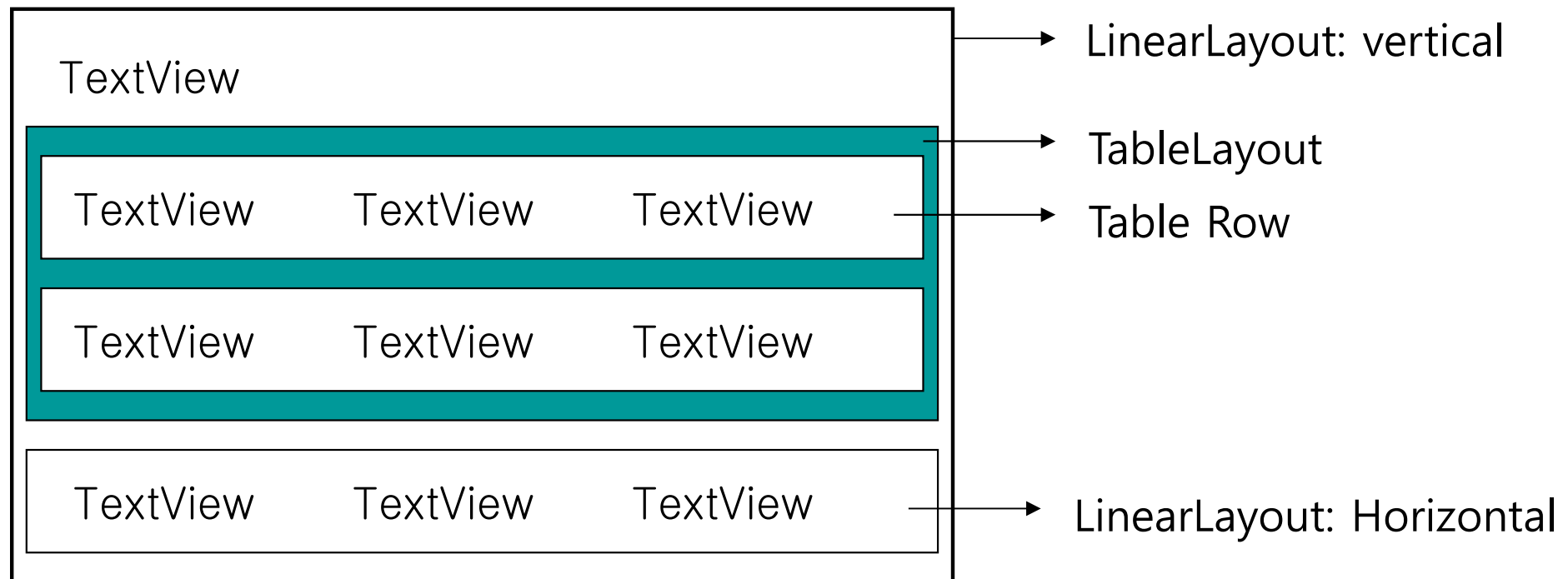
❏ ConstraintLayout

- ◆ 프로젝트 생성 시 기본 레이아웃으로 지정
- ◆ <https://developer.android.com/training/constraint-layout> 참조

레이아웃의 중첩 위젯

레이아웃 안에는 뷰 배치 뿐만 아니라 다른 레이아웃을 중첩하여 배치 가능 → 레이아웃 == 뷰

◆ [Design] 화면의 [Component Tree]에서 관리 용이



XML 활용

- ◆ 레이아웃 및 View를 XML로 지정
- ◆ 지정한 xml 파일을 setContentView(...)에 전달 시 자동으로 객체 변환 및 화면 배치

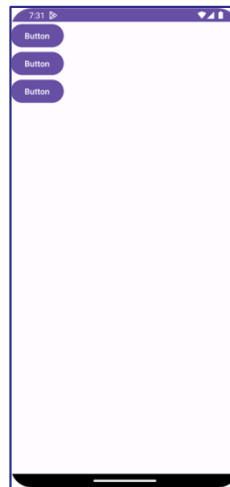
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/linear_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <Button...>

    <Button...>

    <Button...>
</LinearLayout>
```

new_layout.xml



```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.new_layout)
    }
}
```

MainActivity.kt

직접 구현

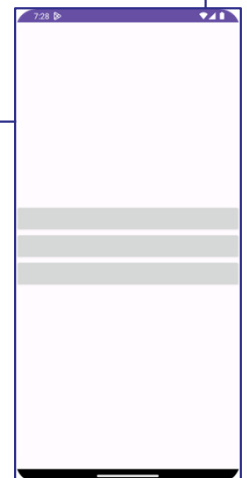
- ◆ 코드로 화면 객체를 직접 구현

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        val layout = LinearLayout(this).apply { this: LinearLayout
            orientation = LinearLayout.VERTICAL
            gravity = Gravity.CENTER
        }
        val button1 = Button(this)
        val button2 = Button(this)
        val button3 = Button(this)
        layout.addView(button1)
        layout.addView(button2)
        layout.addView(button3)

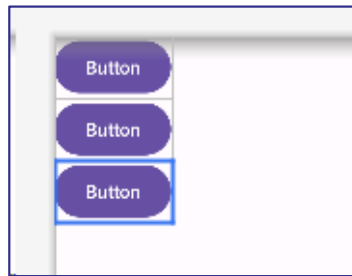
        setContentView(layout)
    }
}
```

MainActivity.kt

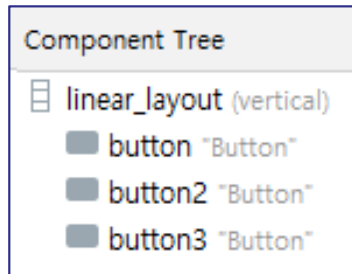


실행 중 레이아웃의 속성 변경

앱 실행 시 레이아웃(또는 View)의 속성 변경



new_layout.xml
(LinearLayout)



```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.new_layout)

        val button : Button = findViewById(R.id.button)
        button.setOnClickListener { it: View!
            val layout : LinearLayout = findViewById(R.id.linear_layout)
            layout.orientation = LinearLayout.HORIZONTAL
        }
    }
}
```

• new_layout.xml을 inflation
및 화면에 배치

• Res ID를 사용하여
객체 준비

• Button 을 눌렀을
경우 동작

- ◆ setContentView(...): 화면에 View 배치, xml 파일명을 전달받을 경우 inflation 수행
 - Inflation: XML을 토대로 실제 객체를 생성하는 작업
- ◆ findViewById(...): Inflation 이후 생성한 객체를 지정한 ID로 찾는 함수

ViewBinding

XML로 지정한 View 객체 확인

- ◆ findViewByld(...) vs. ViewBinding

```
<androidx.constraintlayout.widget.ConstraintLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/textView"
```

ViewBinding

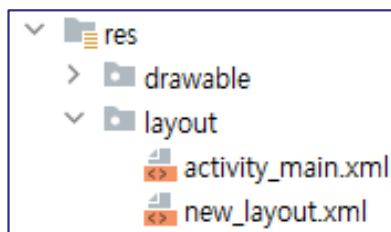
- ◆ XML에서 지정한 ID를 객체명으로 바로 사용

- ◆ 1. 준비

- Build.gradle (Module :app) 편집

```
android { this: BaseAppModuleExtension
    viewBinding { this: ViewBinding
        enable=true
    }
}
```

- ◆ 2. 사용



```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
```

```
val mainBinding : ActivityMainBinding = [ActivityMainBinding.inflate(layoutInflater)]
// val newBinding : NewLayoutBinding = NewLayoutBinding.inflate(layoutInflater)
setContentView(mainBinding.root)
```

```
mainBinding.textView.text = "Hi!!!"
```

```
}
```

• activity_main.xml을 위한
Binding객체
ActivityMainBinding 자동 생성

• 화면에 Binding객체의 최상위 화면
요소를 전달 → root == layout

• XML로 정의한
View 객체 사용

실습 1

다음 형태로 레이아웃 작성

- ◆리니어 레이아웃(vertical)
- ◆버튼 2개 배치

vertical 버튼을 누를 경우

- ◆orientation → vertical

horizontal 버튼을 누를 경우

- ◆orientation → horizontal

ViewBinding을 적용하여 Button을 사용할 것

