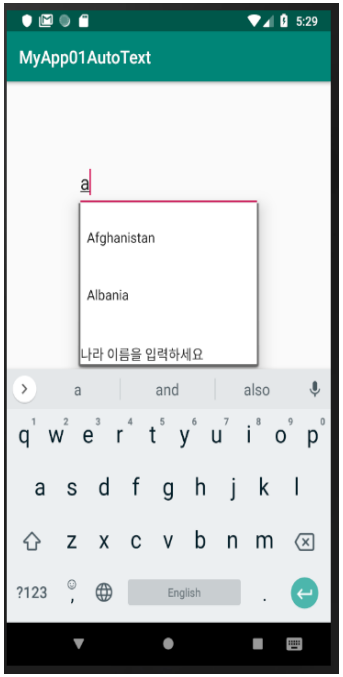
기본 사용법 실습

AutoCompleteTextView

- AutoCompleteTextView
  - Edittext를 상속받아 만들어진 클래스
  - 주요속성
    - **android:completionThreshold (자동완성 글자수)**
    - **android:completionHint (힌트)**



<**AutoCompleteTextView** android:id="*@+id/edit*"

android:layout_width="*match_parent*"

android:layout_height="*wrap_content*"

***android:completionHint =*"나라 이름 중에 일부를 쓰세요"**

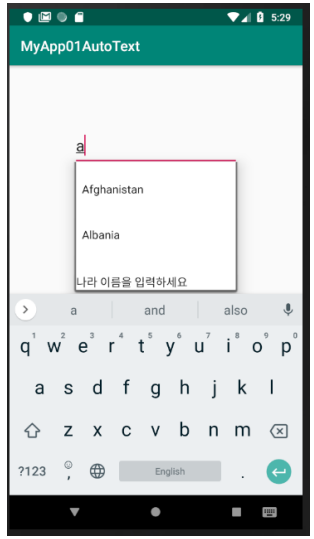
android:completionThreshold="1"

/>

AutoCompleteTextView

- AdapterView

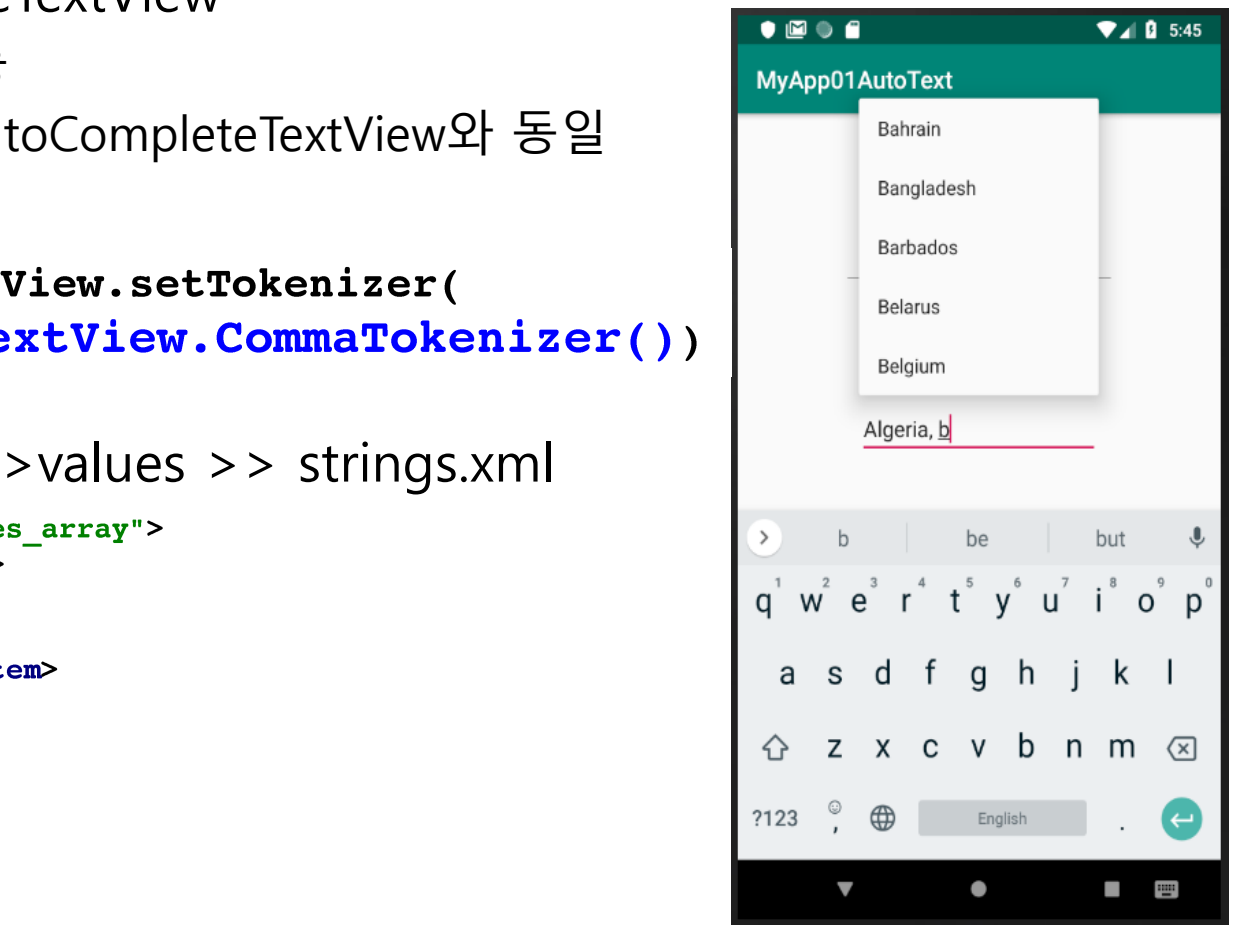
```
val countries = arrayOf(
    "Afghanistan", "Albania", "Algeria", "American Samoa", "Andorra",
    "Angola", "Anguilla", "Antarctica", "Argentina",
    "Armenia", "Aruba", "Australia", "Austria", "Azerbaijan",
    "Bahrain", "Bangladesh", "Barbados", "Belarus", "Belgium")

val adapter = ArrayAdapter<String>(
    this, // Context
    android.R.layout.simple_dropdown_item_1line, // Layout
    countries // Array
)

autoCompleteTextView.setAdapter(adapter)

autoCompleteTextView.setOnItemClickListener{
    parent, view, position, id ->
    val item = parent.getItemAtPosition(position).toString()
    Toast.makeText(this, "선택된 항목 : $item", Toast.LENGTH_SHORT ).show()
}
```



MultiAutoCompleteTextView

- MultiAutoCompleteTextView
  - 여러 개 선택 가능
  - 기본 사용법은 AutoCompleteTextView와 동일
  - 구분자 지정

```
multiAutoCompleteTextView.setTokenizer(
MultiAutoCompleteTextView.CommaTokenizer())
```



Array 생성법 : res>>values >> strings.xml

```
<string-array name="countries_array">
    <item>Afghanistan</item>
    <item>Albania</item>
    <item>Algeria</item>
    <item>American Samoa</item>
    <item>Andorra</item>
    <item>Bahrain</item>
    <item>Bangladesh</item>
    <item>Barbados</item>
    <item>Belarus</item>
    <item>Belgium</item>
</string-array>
```

```
val countries2 = resources.getStringArray(R.array.countries_array)
val adapter2 = ArrayAdapter(this, android.R.layout.simple_list_item_1, countries2)
```

TextWatcher 인터페이스

- TextWatcher
  - 뷰의 값이 변화될 때마다 발생하는 이벤트를 감지하는 인터페이스

➢ ***abstract void afterTextChanged(Editable s)***
- EditText의 텍스트가 변경되면 호출

➢ ***abstract void beforeTextChanged(CharSequence s, int start, int count, int after)***
- start 지점에서 시작되는 count 갯수만큼의 글자들이 after 길이만큼의 글자로 대치되려고 할 때 호출

➢ ***abstract void onTextChanged(CharSequence s, int start, int before, int count)***
- start 지점에서 시작되는 before 갯수만큼의 글자들이 count 갯수만큼의 글자들로 대치되었을 때 호출

TextWatcher 인터페이스

- TextWatcher 사용 예 ➔ 텍스트 입력해야 특정 버튼 활성화

TextWatcher 인터페이스 ➔ object 익명 클래스

```
editText.addTextChangedListener(object : TextWatcher{

    override fun afterTextChanged(s: Editable?) {
        val str = s.toString()
        button.isEnabled = str.isNotEmpty()
    }

    override fun beforeTextChanged(s: CharSequence?, start: Int,
                                   count: Int, after: Int) {
    }

    override fun onTextChanged(s: CharSequence?, start: Int,
                               before: Int, count: Int) {
    }

})
```

EditText : TextWatcher

- Array는 수정할 수 없는(immutable) 객체이므로, 저장 구조를 mutableListOf로 수정해서 적용할 것

```
private val countries= mutableListOf(
    "Afghanistan", "Albania", "Algeria",
"American Samoa", "Andorra", "Angola",
"Anguilla", "Antarctica", "Antigua and Barbuda",
"Argentina","Armenia", "Aruba", "Australia",
"Austria", "Azerbaijan","Bahrain", "Bangladesh",
"Barbados", "Belarus", "Belgium")
```

TextInputLayout / TextInputEditText

- 디자인 라이브러리가 적용된 EditText
- TextInputLayout은 LinearLayout을 상속받아 고안된 레이아웃
  - TextInputEditText를 자식으로 감싸서 사용

```
<com.google.android.material.textfield.TextInputLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/form_username">
    <com.google.android.material.textfield.TextInputEditText
            android:layout_width="match_parent"
            android:layout_height="wrap_content"/>

</com.google.android.material.textfield.TextInputLayout>
```

https://developer.android.com/reference/com/google/android/material/textfield/TextInputLayout

TextInputLayout / TextInputEditText

- 힌트 설정
  - TextInputEditText에 힌트를 입력하고, 포커스가 주어지면 힌트가 TextInputLayout으로 이동해 라벨 메시지로 표시됨

```
android:hint="이메일"
```

- 텍스트 글자수 세기
  - TextInputLayout의 CounterEnabled 속성을 true 설정
  - TextInputLayout의 CounterMaxLength 속성 : 최대 길이
    - 최대 길이를 벗어나면 라벨이나 길이에 대한 색상이 변하게 됨

```
app:counterEnabled="true"
app:counterMaxLength="15"
```

- 에러 메시지 표시하기
  - TextInputLayout의 error 객체에 메시지 추가

```
textInputLayout1.error = "이메일 형식이 올바르지 않습니다."
```

- 비밀번호 보여주는 토글 기능
  - TextInputEditText의 InputType은 textPassword
  - TextInputLayout의 passwordToggleEnabled 속성 true 설정

```
android:inputType="textPassword"
```

```
app:passwordToggleEnabled="true"
```

TextInputLayout / TextInputEditText

- TextWatcher 추가
  - TextInputLayout의 EditText 또는 TextInputEditText에 TextChangedListener 추가함

```
textInputLayout.editText?.addTextChangedListener(object:TextWatcher{

    override fun afterTextChanged(s: Editable?) {
    }

    override fun beforeTextChanged(s: CharSequence?, start: Int,
                                   count: Int, after: Int){
    }

    override fun onTextChanged(s: CharSequence?, start: Int,
                               before: Int, count: Int) {
    }
})
```

Custom component

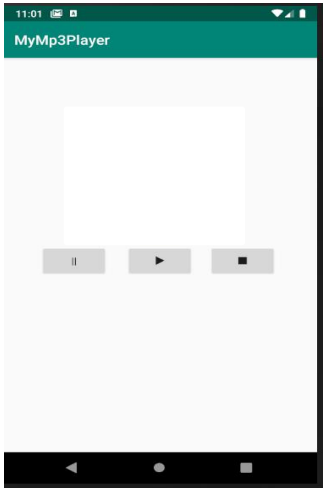
- 개발자가 직접 View 클래스를 상속받아서 만든 위젯
  - View 클래스의 재정의 할 수 있는 콜백 메소드
    - onKeyDown(int, KeyEvent)
    - onKeyUp(int, KeyEvent)
    - onTrackballEvent(MotionEvent)
    - onTouchEvent(MotionEvent)
    - onFocusChanged(boolean, int, Rect)

Custom component

- **기본 위젯**을 상속받아서, 자신만의 위젯으로 작성
  - ImageView를 상속 받고, 자체 Touch 이벤트 처리 기능을 정의
  - ImageViw의 Touch 이벤트 추가
    - **onTouchEvent 함수 오버라이딩**
    - **setOnTouchListener : 뷰 클래스를 생성하지 않을 때 사용**
      - OnTouch(View v, Moton event)
        - » True : onTouchEvent를 호출하지 않음
        - » False : onTouchEvent를 호출 함
          - onTouch (ImageView)
          - onTouchEvent (ImageView)
          - onTouchEvent (Activity)

Custom component

- **인터페이스 정의**
  - 커스텀 위젯의 콜백 함수 정의




```
// 인터페이스를 정의하고, 멤버로 추가 정의
var listener:VolumeListener?=null

public interface VolumeListener{
    public fun onChanged(angle:Float):Unit
}

public fun setVolumeListener(list:VolumeListener):Unit{
    listener = list
}

// 필요시 위젯이 정의한 함수 호출 가능
listener?.onChanged(angle)
```

VolumeControlView

- 터치한 위치의 각도 구하기

```
fun getAngle(x1:Float, y1:Float):Float {
    mx = x1-(width /2.0f)
    my = (height/2.0f)-y1
    return (atan2(mx, my) *180.0f/ PI).toFloat()
}
```

- 뷰 그리기

```
override fun onDraw(canvas: Canvas?) {
    canvas?.rotate(angle,width/2.0f,height/2.0f)
    super.onDraw(canvas)
}
```

* invalidate() : 다시 그리기

MediaPlayer

- 긴 사운드 파일이나 스트림을 재생하기 위해 디자인된 메소드
  - 음악파일이나 큰 동영상 **파일을** 재생하기에 적합함

```
var mPlayer: MediaPlayer?
mPlayer  = MediaPlayer.create(this, R.raw.song);

또는

mPlayer = MediaPlayer();
mPlayer.setDataSource(soundfile_path);
```

준비 : mPlayer.prepare();
재생시작 : mPlayer.start();
일시정지 : mPlayer.pause();
정지 : mPlayer.stop();
메모리 해제 : mPlayer.release();

*setVolume(leftVol, rightVol)
: 0~1 사이의 값 설정
(현재 설정된 volume 값이 최대값)

Kotlin의 Scope functions

- Scope functions
  - 객체의 컨텍스트 내에서 코드 블록을 실행하는 것이 목적인 함수
    - 임시 스코프가 설정되고, 이 스코프 내에서 해당 객체의 이름 없이 멤버 접근이 가능
    - 코드를 간결하고 읽기 쉽게 만드는 역할 수행
  - 5개의 함수 : let, with, run, apply, also

```kotlin
data class Person(var name: String, var age: Int, var city: String)
    fun moveTo(newCity: String) { city = newCity }
    fun incrementAge() { age++ }
}

fun main() {
    val alice = Person("Alice", 20, "Amsterdam")
    println(alice)
    alice.moveTo("London")
    alice.incrementAge()
    println(alice)
}
```

```
Person(name=Alice, age=20, city=Amsterdam)
Person(name=Alice, age=21, city=London)
```

```kotlin
data class Person(var name: String, var age: Int, var city: String)
    fun moveTo(newCity: String) { city = newCity }
    fun incrementAge() { age++ }
}

fun main() {
    Person("Alice", 20, "Amsterdam").let {
        println(it)
        it.moveTo("London")
        it.incrementAge()
        println(it)
    }
}
```

```
Person(name=Alice, age=20, city=Amsterdam)
Person(name=Alice, age=21, city=London)
```

Kotlin의 Scope functions

- Function selection
  - 비슷한 기능을 수행하지만, 객체 참조 방식 및 리턴 값에 따라 선택

Function	Object reference	Return value	Is extension function
`let` ↗	`it`	Lambda result	Yes
`run` ↗	`this`	Lambda result	Yes
`run` ↗	-	Lambda result	No: called without the context object
`with` ↗	`this`	Lambda result	No: takes the context object as an argument.
`apply` ↗	`this`	Context object	Yes
`also` ↗	`it`	Context object	Yes

this 와 it의 차이

- 두 가지 모두 객체를 참조할 때 사용
  - **this**
    - **run, with, apply 에서 객체 참조**
  - **it**
    - **let, also에서 객체 참조**

```
fun main() {
    val str = "Hello"
    // this
    str.run {
        println("The string's length: $length")
        //println("The string's length: ${this.length}") // does the same
    }

    // it
    str.let {
        println("The string's length is ${it.length}")
    }
}
```

Return value

- Scope function이 반환하는 값
  - apply, also: context object
  - let, run, with: lambda result

```
val numberList = mutableListOf<Double>()
numberList.also { println("Populating the list") }
    .apply {
        add(2.71)
        add(3.14)
        add(1.0)
    }
    .also { println("Sorting the list") }
    .sort()
```

```
val numbers = mutableListOf("one", "two", "three")
val countEndsWithE = numbers.run {
    add("four")
    add("five")
    count { it.endsWith("e") }
}
println("There are $countEndsWithE elements that end with e.")
```

```
inline fun <T, R> T.let(block: (T) -> R): R
```

```
inline fun <T, R> with(receiver: T, block: T.() -> R): R
```

```
inline fun <T> T.apply(block: T.() -> Unit): T
```

```
inline fun <T> T.also(block: (T) -> Unit): T
```

```
inline fun <R> run(block: () -> R): R
inline fun <T, R> T.run(block: T.() -> R): R
```

let

- Context object는 argument (it) 로서 이용가능, return value는 lambda result

```
inline fun <T, R> T.let(block: (T) -> R): R
```

  - 하나 이상의 함수를 호출 / null 체크 / local scope에서의 지역 변수 표현

```
val numbers = mutableListOf("one", "two", "three", "four", "five")
numbers.map { it.length }.filter { it > 3 }.let {
    println(it)
    // and more function calls if needed
}
```

```
[5, 4, 4]
```

```
val numbers = listOf("one", "two", "three", "four")
val modifiedFirstItem = numbers.first().let { firstItem ->
    println("The first item of the list is '$firstItem'")
    if (firstItem.length >= 5) firstItem else "!" + firstItem + "!"
}.uppercase()
println("First item after modifications: '$modifiedFirstItem'")
```

```
val str: String? = "Hello"
//processNonNullString(str)       // compilation error: str can be null
val length = str?.let {
    println("let() called on $it")
    processNonNullString(it)      // OK: 'it' is not null inside '?.let { }'
    it.length
}
```

```
let() called on Hello
```

with

- Context object는 receiver (this)로서 이용가능, return value는 lambda result

```
inline fun <T, R> with(receiver: T, block: T.() -> R): R
```

  - 반환된 결과가 필요 없는 함수 호출 / 객체의 속성 설정 및 속성 이나 함수가 값 계산에 사용되는 경우

```
val numbers = mutableListOf("one", "two", "three")
with(numbers) {
    println("'with' is called with argument $this")
    println("It contains $size elements")
}
```

```
'with' is called with argument [one, two, three]
It contains 3 elements
```

```
val numbers = mutableListOf("one", "two", "three")
val firstAndLast = with(numbers) {
    "The first element is ${first()}," +
    " the last element is ${last()}"
}
println(firstAndLast)
```

```
The first element is one, the last element is three
```

run

- Context object는 receiver (this)로서 이용가능, return value는 lambda result

```
inline fun <T, R> T.run(block: T.() -> R): R
```

```
inline fun <R> run(block: () -> R): R
```

  - 객체를 초기화 하고, 반환값 계산 / 식이 필요한 여러 문장의 블록을 실행

```
val service = MultiportService("https://example.kotlinlang.org", 80)

val result = service.run {
    port = 8080
    query(prepareRequest() + " to port $port")
}

// the same code written with let() function:
val letResult = service.let {
    it.port = 8080
    it.query(it.prepareRequest() + " to port ${it.port}")
}
```

```
val hexNumberRegex = run {
    val digits = "0-9"
    val hexDigits = "A-Fa-f"
    val sign = "+-"

    Regex("[$sign]?[$digits$hexDigits]+")
}

for (match in hexNumberRegex.findAll("+123 -FFFF !%*& 88 XYZ")) {
    println(match.value)
}
```

apply

- Context object는 receiver (this)로서 이용가능, return value는 lambda result

```
inline fun <T> T.apply(block: T.() -> Unit): T
```

  - **객체의 초기화**

```
val adam = Person("Adam").apply {
    age = 32
    city = "London"
}
println(adam)
```

also

- Context object는 argument (it) 로서 이용가능, return value는 object itself

```
inline fun <T> T.also(block: (T) -> Unit): T
```

  - Context object를 인수로서 사용하는 동작
    - 객체의 속성을 변경하지 않고 사용하는 경우

```
val numbers = mutableListOf("one", "two", "three")
numbers
    .also { println("The list elements before adding new one: $it") }
    .add("four")
```

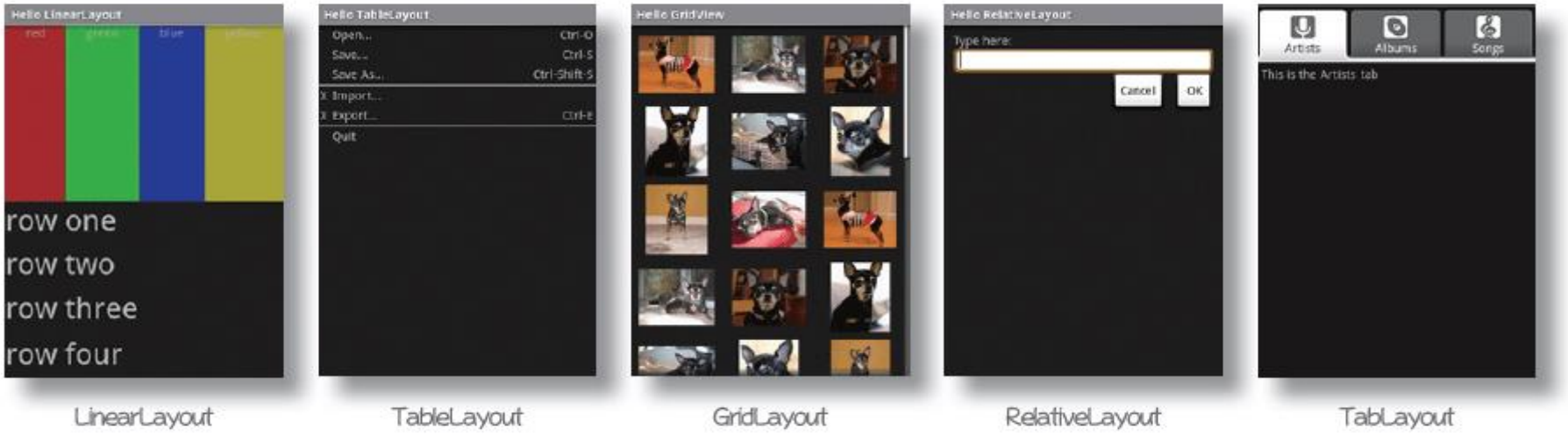
Layout

Layout

- 위젯을 목적에 맞게 배치하고 깔끔하게 정리할 수 있게 하는 역할
  - ViewGroup 클래스로 부터 파생된 클래스로 위젯 배치하는 역할
  - 자주 사용되는 속성
    - orientation
      - 리니어 레이아웃에서 쓰이는 형태로 수직 또는 수평방향 설정
    - gravity
      - 레이아웃 안에 배치할 위젯의 정렬 방향을 좌측,우측,중앙 등으로 설정
    - padding
      - 레이아웃 안에 배치할 위젯의 여백을 설정
    - layout_weight
      - 레이아웃이 전체 화면에서 차지하는 공간의 가중치 값을 설정
    - baselineAligned
      - 레이아웃 안에 배치할 위젯들을 아랫줄에 맞춰 보기 좋게 정렬

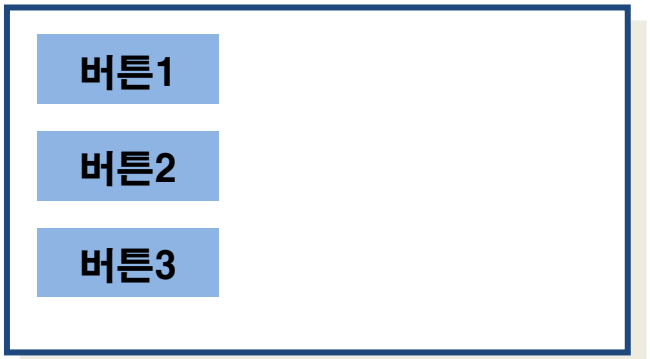
Layout의 종류

- LinearLayout : 자식들을 수평이나 수직으로 배치
- TableLayout : 자식들을 테이블 형태로 배치
- GridLayout : 자식들을 바둑판 모양으로 배치
- RelativeLayout : 자식들을 부모나 다른 자식에 상대적으로 배치
- TabLayout : 탭을 이용하여 겹쳐진 자식 중에서 하나를 선택
- FrameLayout : 모든 자식들을 겹치게 배치
- ConstraintLayout : 내부에 중첩된 레이아웃을 사용하지 않는 배치

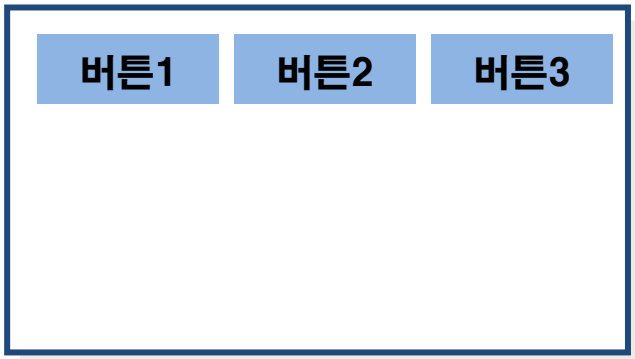


LinearLayout

- 차일드 뷰를 수평, 수직으로 일렬 배치하는 레이아웃으로, 가장 단순하면서 직관적임
  - **https://developer.android.com/guide/topics/ui/layout/linear.html**
- **orientation**
  - 뷰의 배치 방향을 결정하는 속성
  - **vertical** : 차일드를 위에서 아래로 수직으로 배열
  - **horizontal** : 차일드를 왼쪽에서 오른쪽으로 수평 배열 (default)



[vertical]



[horizontal]

LinearLayout

- **gravity**
  - **내용물의 위치를 지정하며, 수평, 수직 방향에 대해 각각 정렬 방식을 지정**할 수 있다.
  - 두 속성을 같이 지정할 때는"|"연산자를 이용하며, 이 때 연산자 양쪽으로 공백이 전혀 없어야 함

상수	값	설명
center_horizontal	0x01	수평으로 중앙에 배치한다.
left	0x03	컨테이너의 왼쪽에 배치하며, 크기는 바뀌지 않는다.
right	0x05	컨테이너의 오른쪽에 배치한다.
fill_horizontal	0x07	수평 방향으로 가득 채운다.
center_vertical	0x10	수직으로 중앙에 배치한다.
top	0x30	컨테이너의 상단에 배치하며, 크기는 바뀌지 않는다.
bottom	0x50	컨테이너의 하단에 배치한다.
fill_vertical	0x70	수직 방향으로 가득 채운다.
center	0x11	수평으로나 수직으로 중앙에 배치한다.
fill	0x77	컨테이너에 가득 채우도록 수직, 수평 크기를 확장한다.

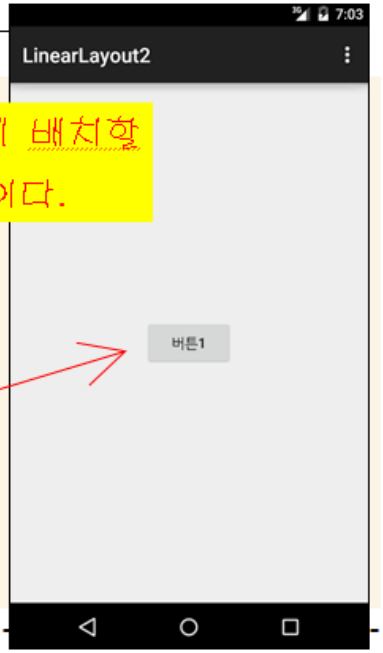
LinearLayout

❖ **gravity**

main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    >
<Button  android:id="@+id/button01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="버튼1"
    />
</LinearLayout>
```

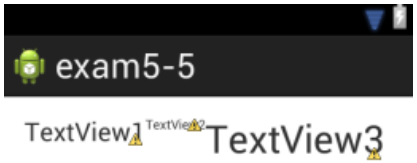
"자식뷰를 중앙에 배치할 것!"이라는 의미이다.



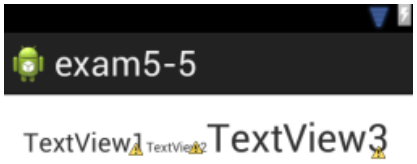
LinearLayout

- **baselineAligned 속성**

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    android:baselineAligned="false"
```




```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    android:baselineAligned="true"
```



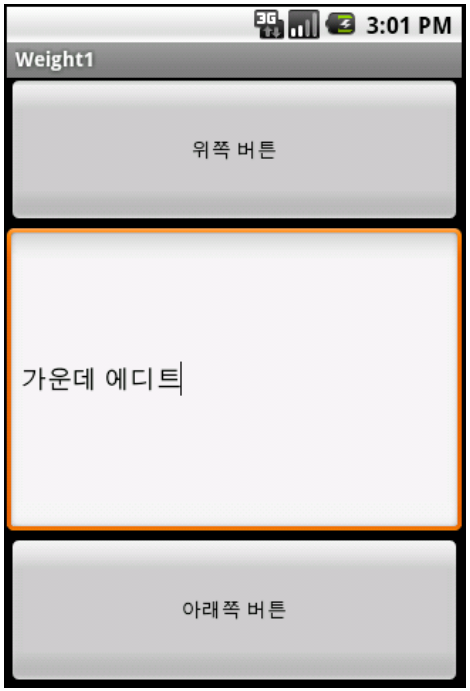
LinearLayout

❖ layout_weight

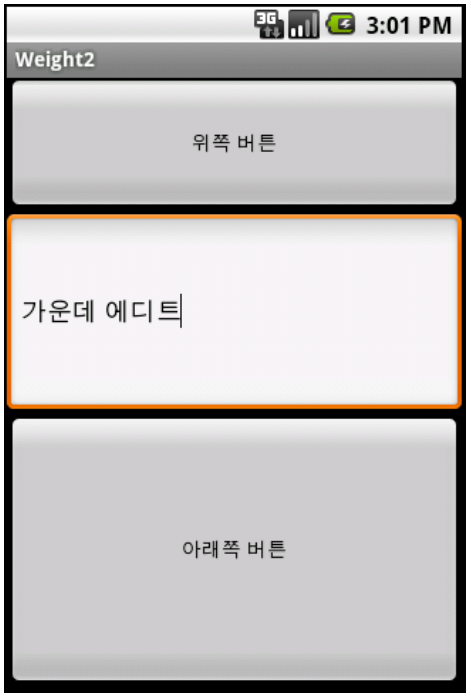
- 중요도에 따라 차일드의 크기를 균등 분할한다.
- 중요도가 0이면 자신의 고유한 크기만큼, 1 이상이면 형제 뷰와의 비율에 따라 부모의 영역을 균등하게 배분한다.

Layout/weight1.xml ~ weight2.xml

```
<?xml version="1.0" encoding="utf-8">
<LinearLayout xmlns:android=http://schemas.android.com/apk/res/android
      android:orientation="vertical"
      android:layout_width="match_parent"
      android:layout_height="match_parent">
<Button
          android:layout_width="match_parent"
          android:layout_height="0dp"
          android:text="위쪽 버튼"
          android:layout_weight="1"/>
<EditText
          android:layout_width="match_parent"
          android:layout_height="0dp"
          android:text="가운데 에디트"
          android:layout_weight="3"/>
<Button
          android:layout_width="match_parent"
          android:layout_height="0dp"
          android:text="아래쪽 버튼"
          android:layout_weight="1"/>
</LinearLayout>
```



[weight1]



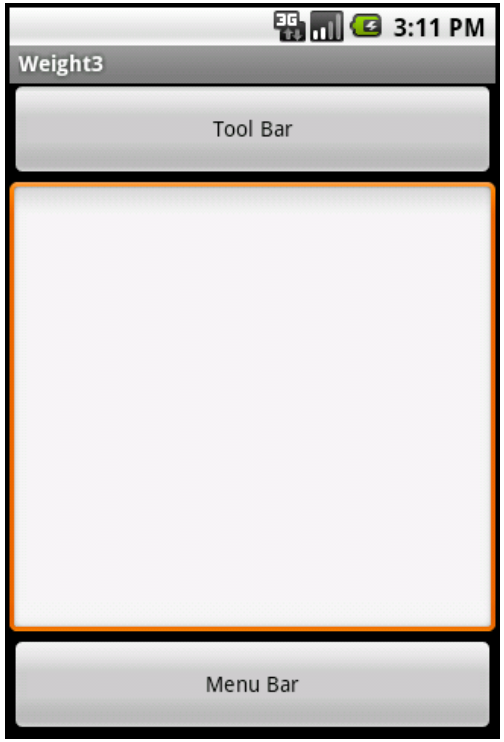
[weight2]

LinearLayout

❖ layout_weight

Layout/weight3.xml

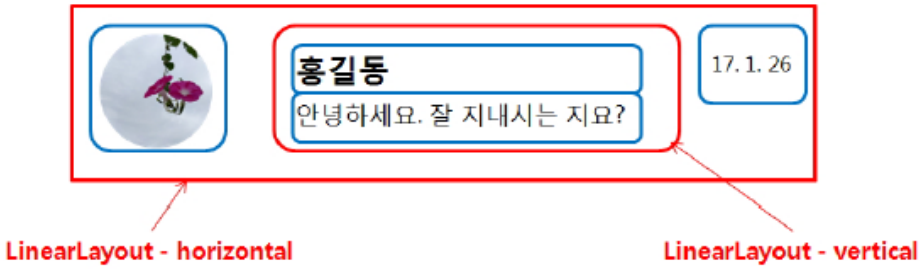
```
<?xml version="1.0" encoding="utf-8">
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">
  <Button
    android:layout_width="match_parent"
    android:layout_height="50dp"
    android:text="Tool Bar"/>
  <EditText
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1"/>
  <Button
    android:layout_width="match_parent"
    android:layout_height="50dp"
    android:text="Menu Bar"/>
</LinearLayout>
```

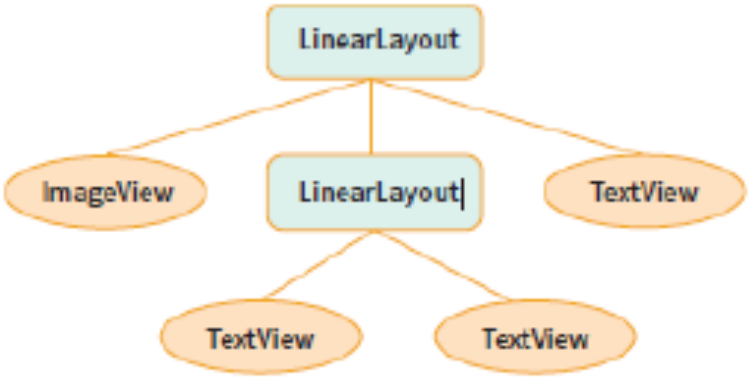


[weight3]

Layout 중첩

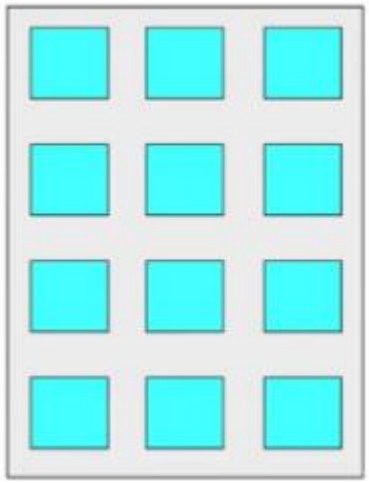
- 레이아웃 중첩
  - 레이아웃은 뷰의 컨테이너이므로 View로부터 파생된 모든 객체를 레이아웃 안에 놓을 수 있으며, 레이아웃 자체도 View의 파생 클래스이므로 레이아웃끼리 중첩시킬 수 있다.





GridLayout

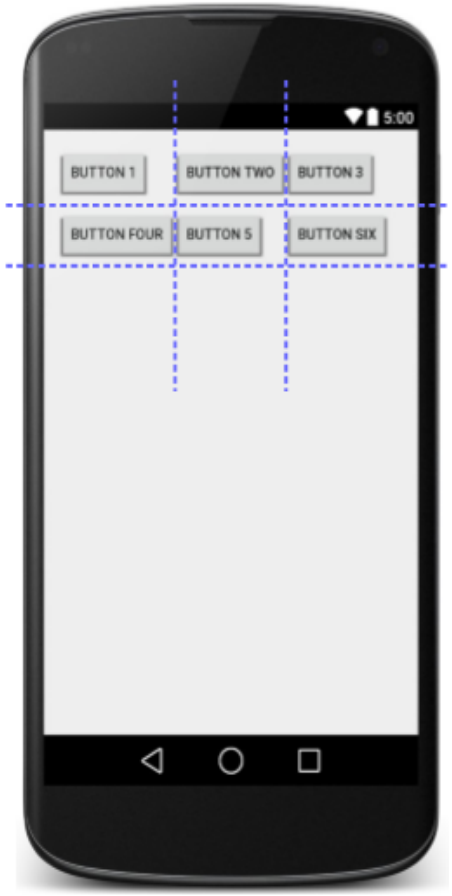
- 행(row) 와 열(columns)으로 위젯을 배치
  - Orientation 속성으로 행 우선 또는 열 우선을 정의함
  - 디폴트로 행과 열은 같은 크기로 만들어짐



4행 3열

GridLayout

```
<GridLayout ...
        android:rowCount="2"
        android:columnCount="3"
        tools:context=".MainActivity">
    <Button ... android:text="Button 1" />
    <Button ... android:text="Button Two" />
    <Button ... android:text="Button 3" />
    <Button ... android:text="Button Four" />
    <Button ... android:text="Button 5" />
    <Button ... android:text="Button Six" />
</GridLayout>
```

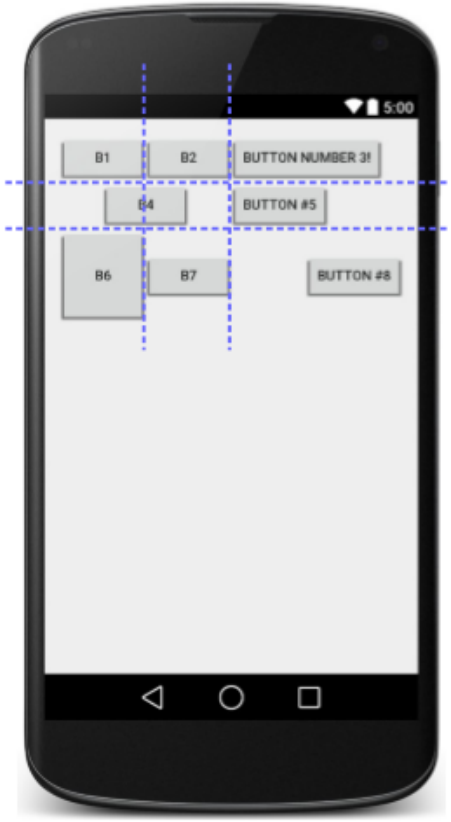


GridLayout

```
<GridLayout ...
        android:rowCount="2"
        android:columnCount="3"
        android:orientation="vertical">
    <Button ... android:text="Button 1" />
    <Button ... android:text="Button Two" />
    <Button ... android:text="Button 3" />
    <Button ... android:text="Button Four" />
    <Button ... android:text="Button 5"
                android:layout_row="1"
                android:layout_column="2" />
    <Button ... android:text="Button Six"
                android:layout_row="0"
                android:layout_column="2" />
```

GridLayout

```
<GridLayout ...
        android:rowCount="2"
        android:columnCount="3">
    <Button ... android:text="B1" />
    <Button ... android:text="B2" />
    <Button ... android:text="Button Number 3!" />
    <Button ... android:text="B4"
       android:layout_columnSpan="2"
       android:layout_gravity="center" />
    <Button ... android:text="B5" />
    <Button ... android:text="B6"
       android:layout_paddingTop="40dp"
       android:layout_paddingBottom="40dp" />
    <Button ... android:text="B7" />
    <Button ... android:text="Button #8"
       android:layout_gravity="right" />
```



FrameLayout

❖ 프레임 레이아웃

- 위젯을 배치하는 특정 규칙이 없이, 레이아웃 내의 위젯들은 왼쪽 상단부터 겹쳐서 출력함

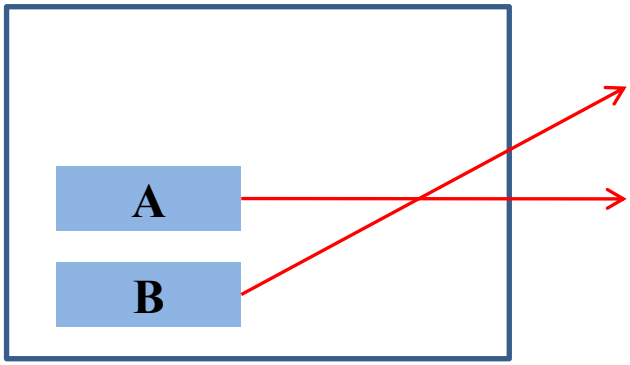
Layout/frame.xml

```
<?xml version="1.0" encoding="utf-8">
<FrameLayout xmlns:android=http://schemas.android.com/apk/res/android
        android:layout_width="match_parent"
        android:layout_height="match_parent">
<Button
          android:id="@+id/btn"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="Push Button"/>
<ImabeView
          android:id="@+id/img"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:src="@drawable/pride"/>
</FrameLayout>
```

RelativeLayout

❖ RelativeLayout

- 위젯과 부모와의 위치 관계 또는 위젯끼리의 관계를 지정함으로써 뷰를 배치
- 위젯끼리의 관계 지정을 위하여 기준이 되는 위젯에 **id**를 반드시 지정해야 함
- 특정 뷰가 다른 뷰의 위치에 종속적일 때 기준이 되는 뷰를 먼저 정의해야 함



[원하는 배치]

```
<RelativeLayout>
<B
   android:id="@+id/b"
/>
<A
   layout_above="@id/b"
/>
</RelativeLayout>
```

[XML 문서]

RelativeLayout

❖ 배치 속성

속성	설명
layout_above	만약 true이면 현재 뷰의 하단을 기준 뷰의 위에 일치시킨다.
layout_below	현재 뷰의 상단을 기준 뷰의 하단에 위치시킨다.
layout_centerHorizontal	수평으로 현재 뷰의 중심을 부모와 일치시킨다.
layout_centerInParent	부모의 중심점에 현재 뷰를 위치시킨다.
layout_centerVertical	수직으로 현재 뷰의 중심을 부모와 일치시킨다.
layout_toLeftOf	현재 뷰의 우측단을 기준 뷰의 좌측단에 위치시킨다.
layout_toRightOf	현재 뷰의 좌측단을 기준 뷰의 우측단에 위치시킨다.

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/address"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:text="주소를 입력하세요" />

    <EditText
        android:id="@+id/input"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@android:drawable/editbox_background"
        android:layout_below="@id/address" />

    <Button
        android:id="@+id/cancel"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/input"
        android:layout_alignParentRight="true"
        android:layout_marginLeft="10dip"
        android:text="취소" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toLeftOf="@id/cancel"
        android:layout_alignTop="@id/cancel"
        android:text="확인" />

</RelativeLayout>
```

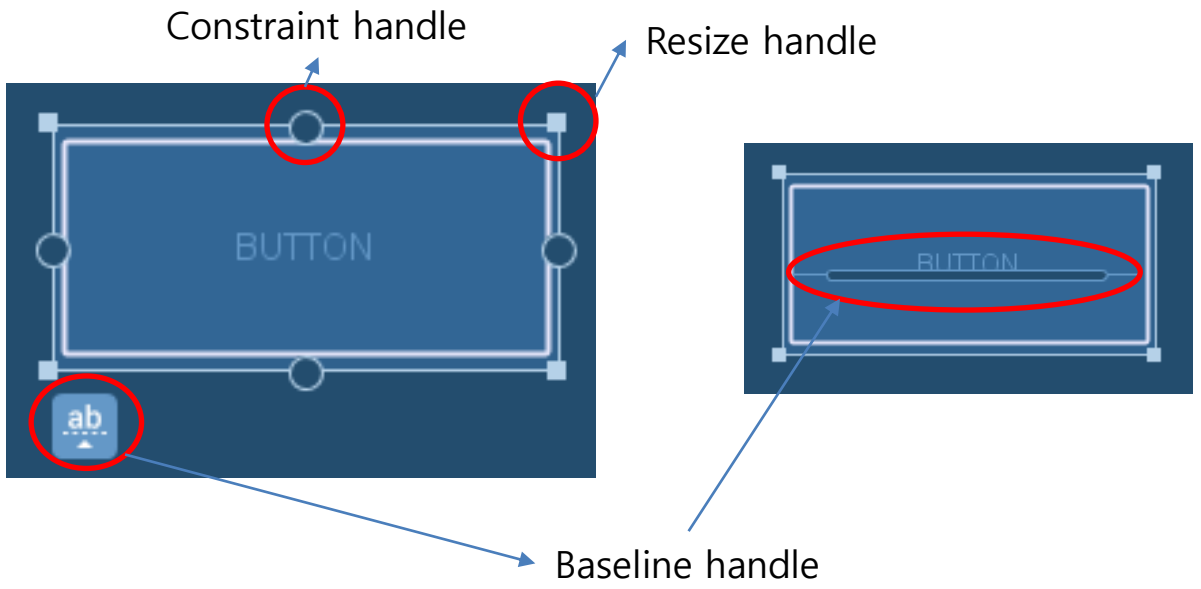
address 아래에 배치

input 아래에 배치

cancel의 왼쪽에 배치

ConstraintLayout

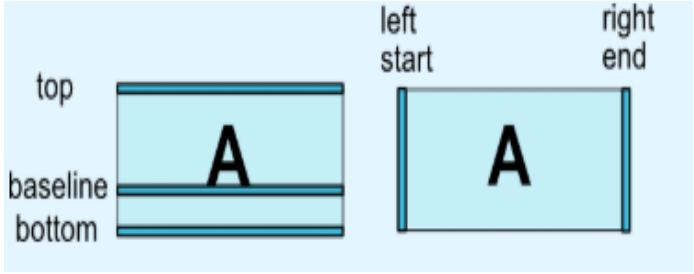
- 위젯을 유연하게 배치하고, 크기를 지정할 수 있는 레이아웃
  - 평면 뷰 계층(중첩된 뷰 그룹 없이)으로 크고 복잡한 레이아웃을 만들 수 있음
  - Android Studio의 레이아웃 편집기에서 사용하기 쉬움 ➔ XML 편집 안해도 됨
    - https://developer.android.com/reference/androidx/constraintlayout/widget/ConstraintLayout
    - https://developer.android.com/training/constraint-layout?hl=ko



ConstraintLayout

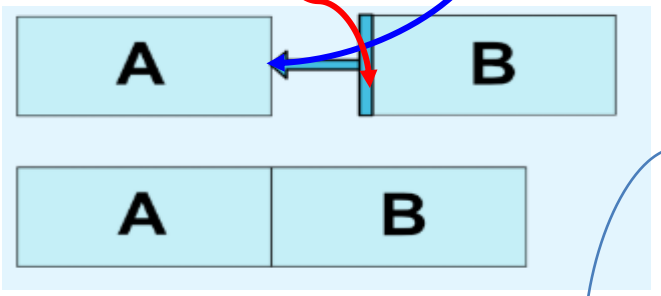
❖ 위치 지정

- 가로 축 : left, right, start, end의 상대적인 위치 지정
- 세로 축 : top, bottom, baseline의 상대적 위치 지정



- layout_constraintLeft_toLeftOf
- layout_constraintLeft_toRightOf
- layout_constraintRight_toLeftOf
- layout_constraintRight_toRightOf
- layout_constraintTop_toTopOf
- layout_constraintTop_toBottomOf
- layout_constraintBottom_toTopOf
- layout_constraintBottom_toBottomOf
- layout_constraintBaseline_toBaselineOf
- layout_constraintStart_toEndOf
- layout_constraintStart_toStartOf
- layout_constraintEnd_toStartOf
- layout_constraintEnd_toEndOf

```
<Button android:id="@+id/buttonA" ... />
        <Button android:id="@+id/buttonB" ...
                app:layout_constraintLeft_toRightOf="@+id/buttonA" />
```

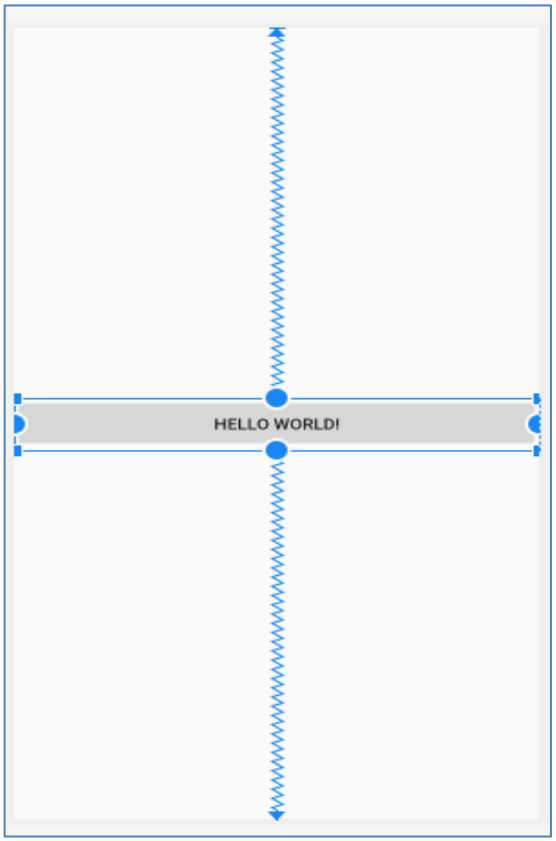


*** 다른위젯의 id 또는 “parent”(parent container) 참조**

ConstraintLayout

- 뷰의 크기
  - layout_width/height : match_constraint

```
<Button
    android:id="@+id/button"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

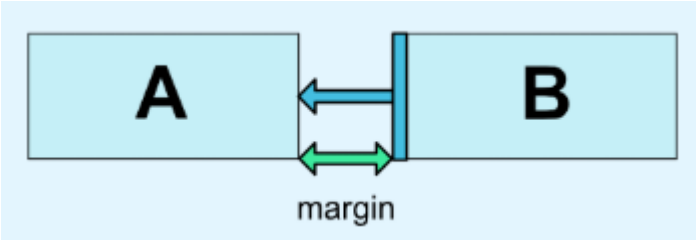


ConstraintLayout

❖ Margins

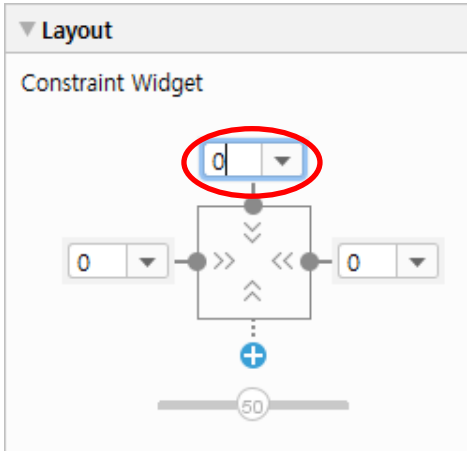
- 뷰와 뷰 사이의 간격을 표현하기 위해 margin 설정

```
• android:layout_marginStart
• android:layout_marginEnd
• android:layout_marginLeft
• android:layout_marginTop
• android:layout_marginRight
• android:layout_marginBottom
```



- 상대뷰가 View.Gone 상태일 때 따로 margin 설정

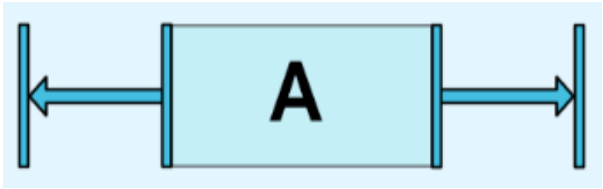
```
• layout_goneMarginStart
• layout_goneMarginEnd
• layout_goneMarginLeft
• layout_goneMarginTop
• layout_goneMarginRight
• layout_goneMarginBottom
```



ConstraintLayout

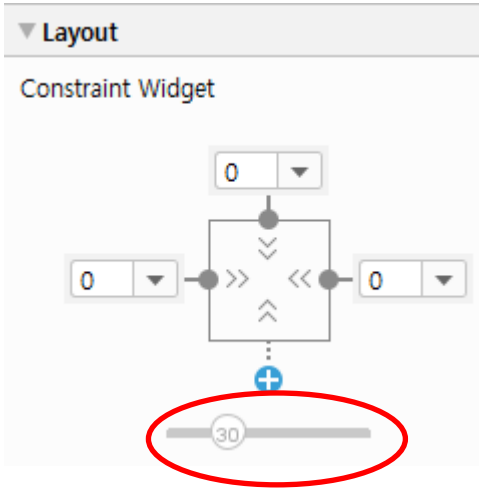
- Centering positioning
  - 상반된 조건 둘을 같이 사용하면 중앙에 위치 시킴

```
<android.support.constraint.ConstraintLayout ...>
            <Button android:id="@+id/button" ...
                app:layout_constraintLeft_toLeftOf="parent"
                app:layout_constraintRight_toRightOf="parent/>
        </>
```



- Bias (치우침)
  - layout_constraintHorizontal_bias
  - layout_constraintVertical_bias

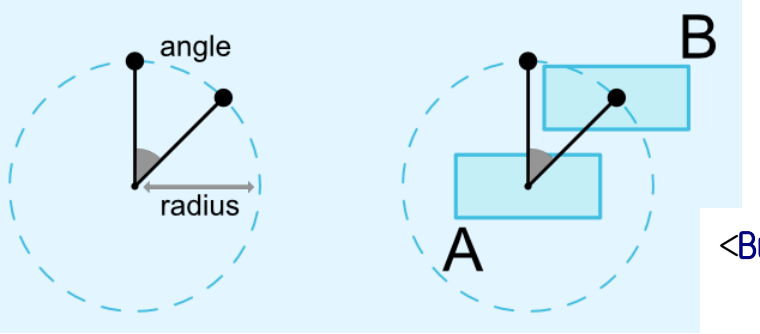

```
<android.support.constraint.ConstraintLayout ...>
            <Button android:id="@+id/button" ...
                app:layout_constraintHorizontal_bias="0.3"
                app:layout_constraintLeft_toLeftOf="parent"
                app:layout_constraintRight_toRightOf="parent/>
        </>
```



ConstraintLayout

- Circular positioning
  - 상대 위젯의 중심, 각도, 반지름을 이용하여 위젯의 중심을 결정

- **layout_constraintCircle** : references another widget id
- **layout_constraintCircleRadius** : the distance to the other widget center
- **layout_constraintCircleAngle** : which angle the widget should be at (in degrees, from 0 to 360)




```
<Button android:id="@+id/buttonA" ... />
  <Button android:id="@+id/buttonB" ...
      app:layout_constraintCircle="@+id/buttonA"
      app:layout_constraintCircleRadius="100dp"
      app:layout_constraintCircleAngle="45" />
```

```
<Button
    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button"
    app:layout_constraintBottom_toTopOf="@+id/button1"
    app:layout_constraintStart_toStartOf="@+id/button1"
    app:layout_constraintCircle="@id/button1"
    app:layout_constraintCircleAngle="45"
    app:layout_constraintCircleRadius="100dp" />
```

ConstraintLayout

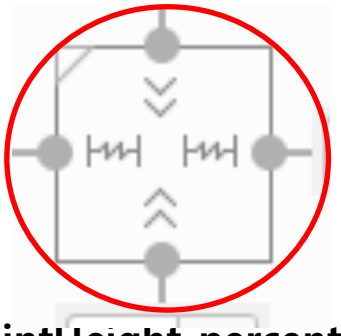
- Dimensions constraints
  - **Minimum dimensions on ConstraintLayout**
    - `android:minWidth` set the minimum width for the layout
    - `android:minHeight` set the minimum height for the layout
    - `android:maxWidth` set the maximum width for the layout
    - `android:maxHeight` set the maximum height for the layout
  - **Widgets dimension constraints**
    - **android:layout_width / android:layout_height**

      -. 고정 길이

      -. wrap_content

      -. 0dp → match_constraint

      -. match_parent 사용하지 말 것



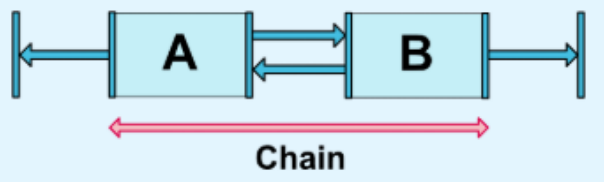
    - **layout_constraintWidth_percent/layout_constraintHeight_percent**

      -. Match_constraint일 때, 적용할 비율 (예, 0.7 : 전체 크기의 70% 크기)

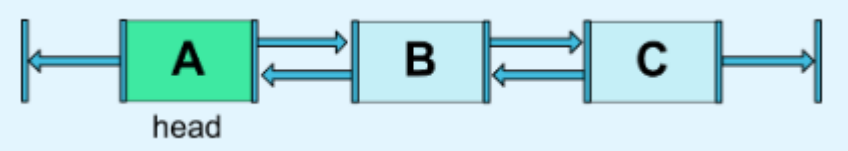
ConstraintLayout

❖ Chains

- 가로 또는 세로 축을 기준으로 그룹을 생성
- Creating a chain



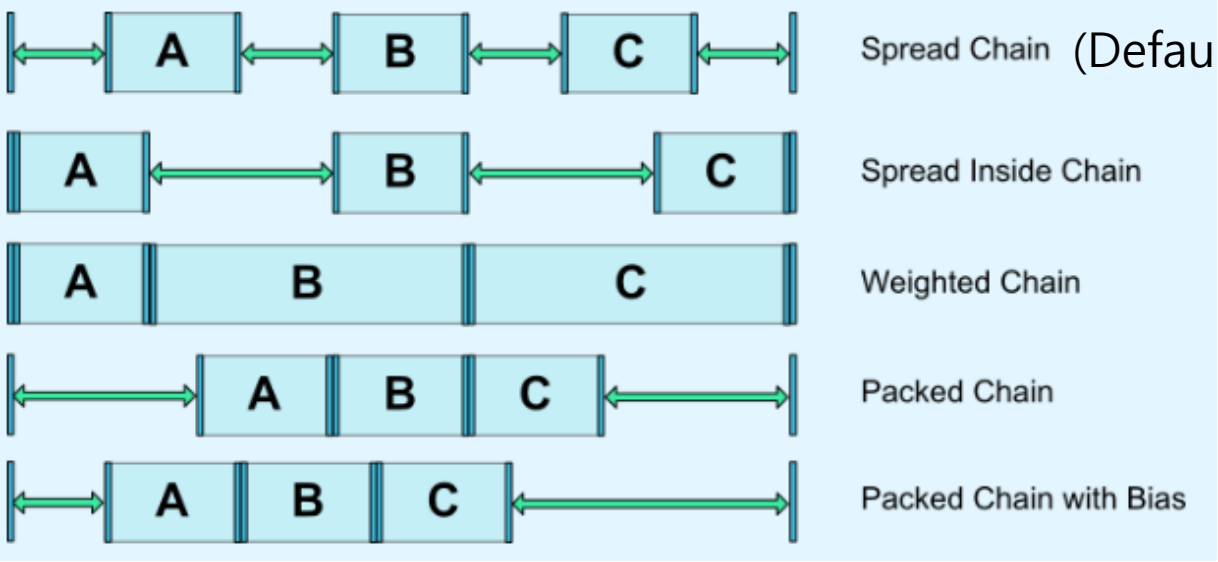
- Chain heads



ConstraintLayout

❖ Chains

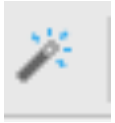
- **Chain style**
  - **layout_constraintHorizontal_chainStyle / layout_constraintVertical_chainStyle**
  - **layout_constraintHorizontal_weight / layout_constraintVertical_weight**
  - 체인중 헤더의 스타일을 변경



ConstraintLayout

- Autoconnect ON/OFF 
  - 레이아웃 내 위젯을 배치할 때 자동으로 이웃한 위젯이나 화면 경계와의 관계를 자동으로 지정해 줌

- Constraint 삭제 
  - 현재 설정되어 있는 관계들을 모두 제거함
  - 위젯이 선택된 상태에서 누르면 해당 위젯의 관계만 제거됨

- Infer constraints 
  - 레이아웃 내 배치된 전체 위젯의 현재 상태를 기반으로 관계를 지정함

Resource ID

❖ Resource id 획득하는 방법

R.id.hat

```
val imgID = resources.getIdentifier( “hat” , "id", packageName)
```

R.drawable.hat

```
val imgR = resources.getIdentifier( “hat” ,  “drawable", packageName)
```

수고하셨습니다.