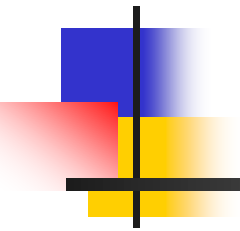
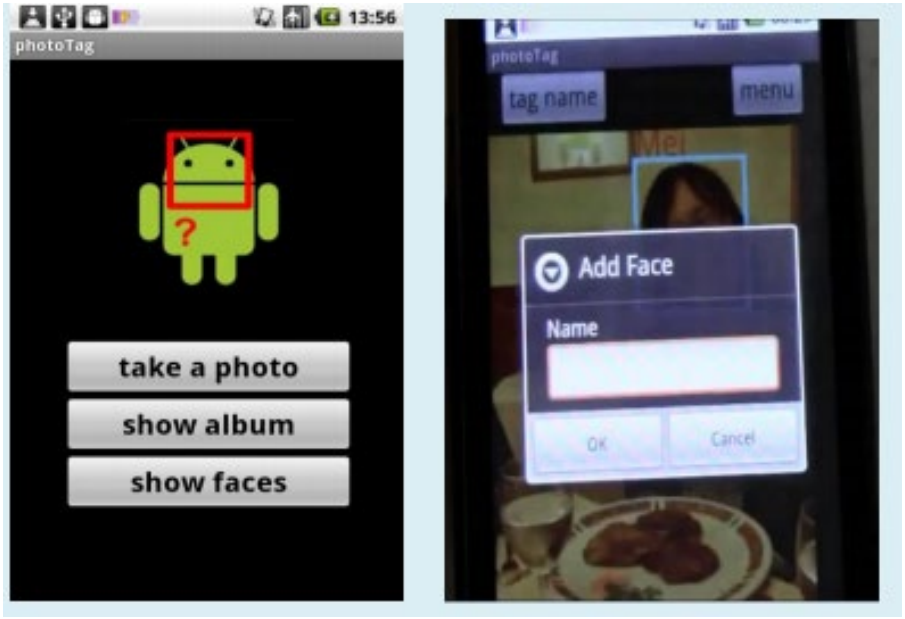


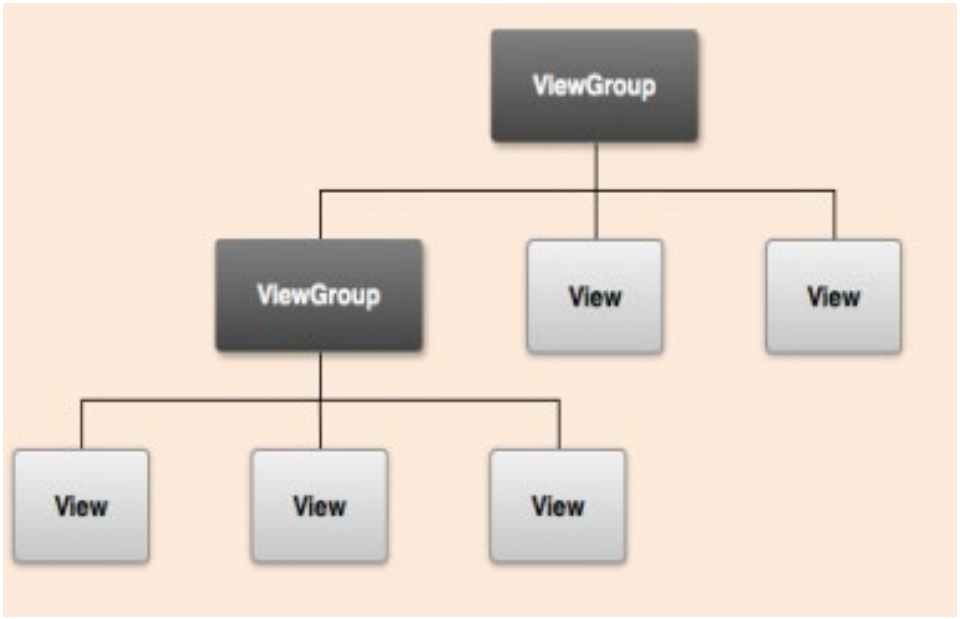
Layout

배 희호 교수
경복대학교
소프트웨어융합과

Android GUI 개념

- GUI = View 및 ViewGroup 객체의 계층 구조
- View = UI 구성 요소(예 : Button, TextView, ImageView,..)
- ViewGroup = View Widget이 정렬되는 방식을 제어하는 Layout이 정의된 Containers





View에 대한 이해는, Android 개발에 대한 이해의 기초이자 시발점

Layout

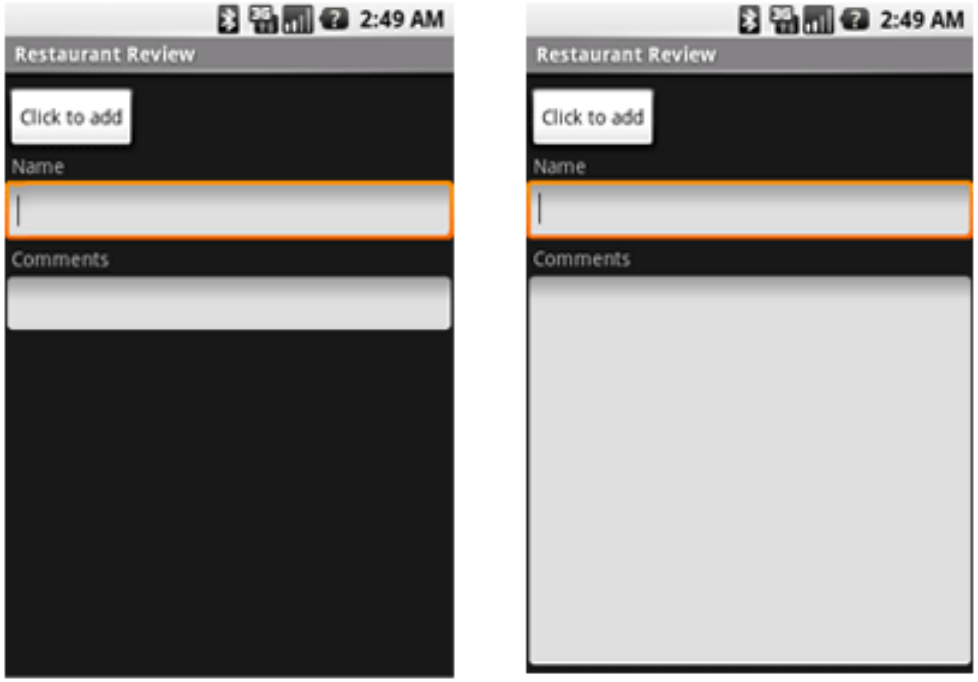
- Layout이란?
  - 사전적 의미

> [명사]
> 책이나 신문, 잡지 따위에서 글이나 그림 따위를 효과적으로 정리하고 배치하는 일
> 정원 따위의 설계를 이르는 말
> 양재에서, 패턴 종이를 배열하는 일

  - Layout이란 뭔가의 구조를 잡고, 어떻게 배열할지, 어떻게 배치할지, 어떻게 설계할지를 정하는 것
  - Layout은 사용자에게 어떻게 보여줄 것인지를 고려해서 UI(User Interface)를 만들고, Widget들을 구성하고 배치하여 좀 더 적합하고 효율적이고 편하게 쓸 수 있도록 해주는 것

Layout

- Layout이란 보여지는 시각물을 보다 간결하게 정리, 배열, 배치하는 효과와 함께 가독성을 높이기 위한 작업 과정을 말함



이와 같이 TextBox의 크기나 위치를 조절할 수 있음

Layout

- 종류

(a) 리니어레이아웃 (b) 렐러티브레이아웃 (c) 테이블레이아웃 (d) 그리드레이아웃 (e) 프레임레이아웃

그림 5-1 레이아웃의 종류

Layout

Layout	설 명
LinearLayout	✓ 박스(Box) 모델 ✓ 사각형 영역들을 이용해 화면을 구성하는 방법 ✓ 표준 JAVA의 BoxLayout과 유사 ✓ 왼쪽 위부터 아래쪽 또는 오른쪽으로 차례로 배치
RelativeLayout	✓ 규칙(Rule) 기반 모델 ✓ 부모 Container나 다른 View와의 상대적 위치를 이용해 화면을 구성하는 방법 ✓ Widget 자신이 속한 Layout의 상하좌우의 위치를 지정하여 배치
FrameLayout	✓ 기본 단위 모델 ✓ 하나의 View만 보여주는 방법 ✓ 가장 단순하지만 여러 개의 View를 추가하는 경우 중첩시킬 수 있으므로 View를 중첩한 후 각 View를 전환하여 보여주는 방식으로 사용할 때 유용함 ✓ Widget들을 왼쪽 위에 일률적으로 겹쳐서 배치하여 중복해서 보이는 효과를 냄

Layout

Layout	설 명
TableLayout	✓ 격자(Grid) 모델 ✓ 격자 모양의 배열을 이용하여 화면을 구성하는 방법 ✓ HTML에서 많이 사용하는 정렬 방식과 유사하여 실용적 임 ✓ Widget을 행과 열의 개수를 지정한 Table 형태로 배열
GridLayout	✓ TableLayout과 비슷하지만, 행 또는 열을 확장하여 다양하게 배치할 때 더 편리
ScrollView	✓ Scroll이 가능한 Container ✓ View 또는 ViewGroup이 들어갈 수 있으며 화면 영역을 넘어갈 때 Scroll 기능 제공

Layout

- Layout에 따라 View를 추가하는 방식

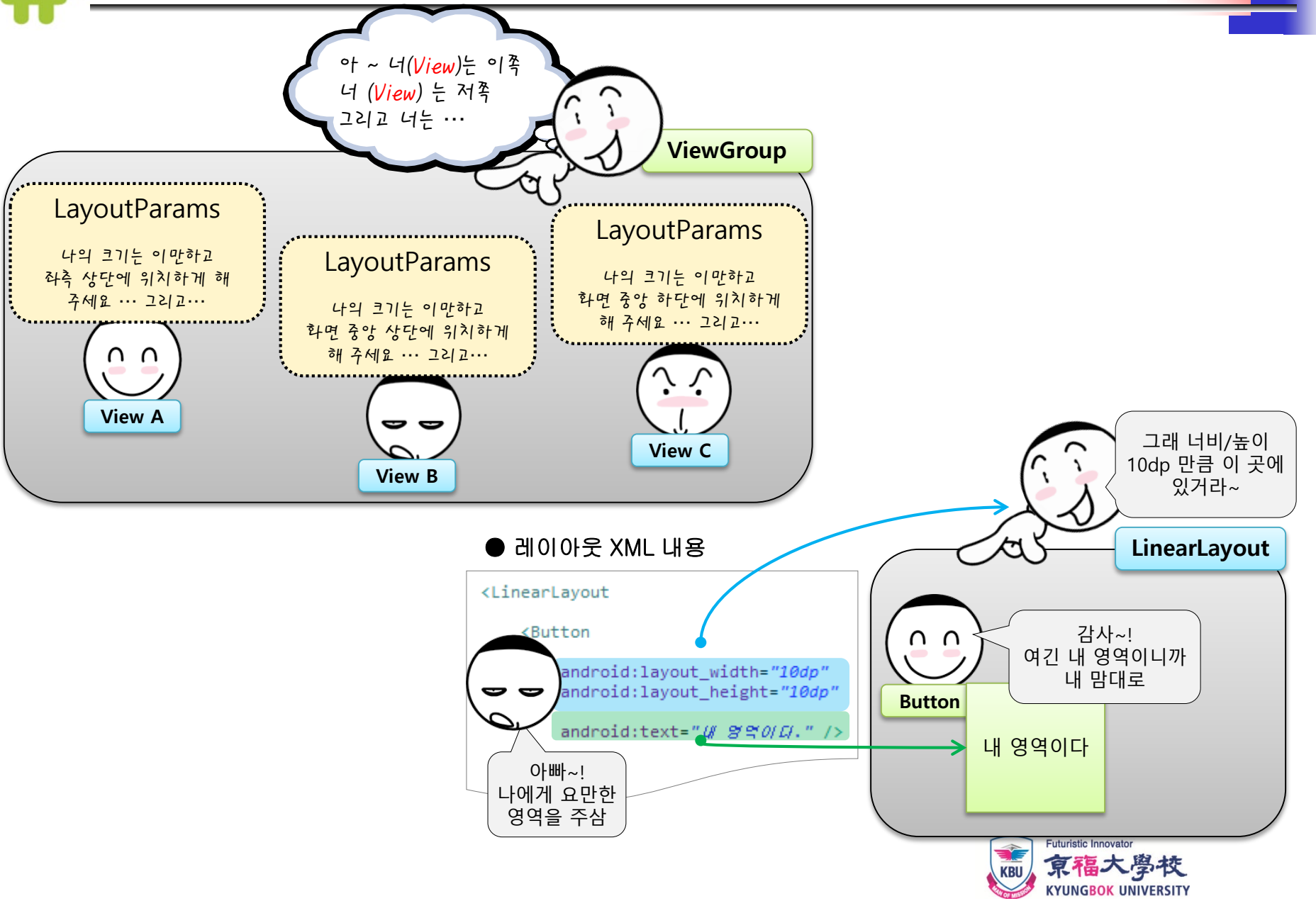
Layout

- Layout은 ViewGroup 클래스로부터 상속받으며 내부에 무엇을 담는 용도로 사용
- Layout 중에서 가장 많이 사용되는 것은 LinearLayout

Layout 계층도

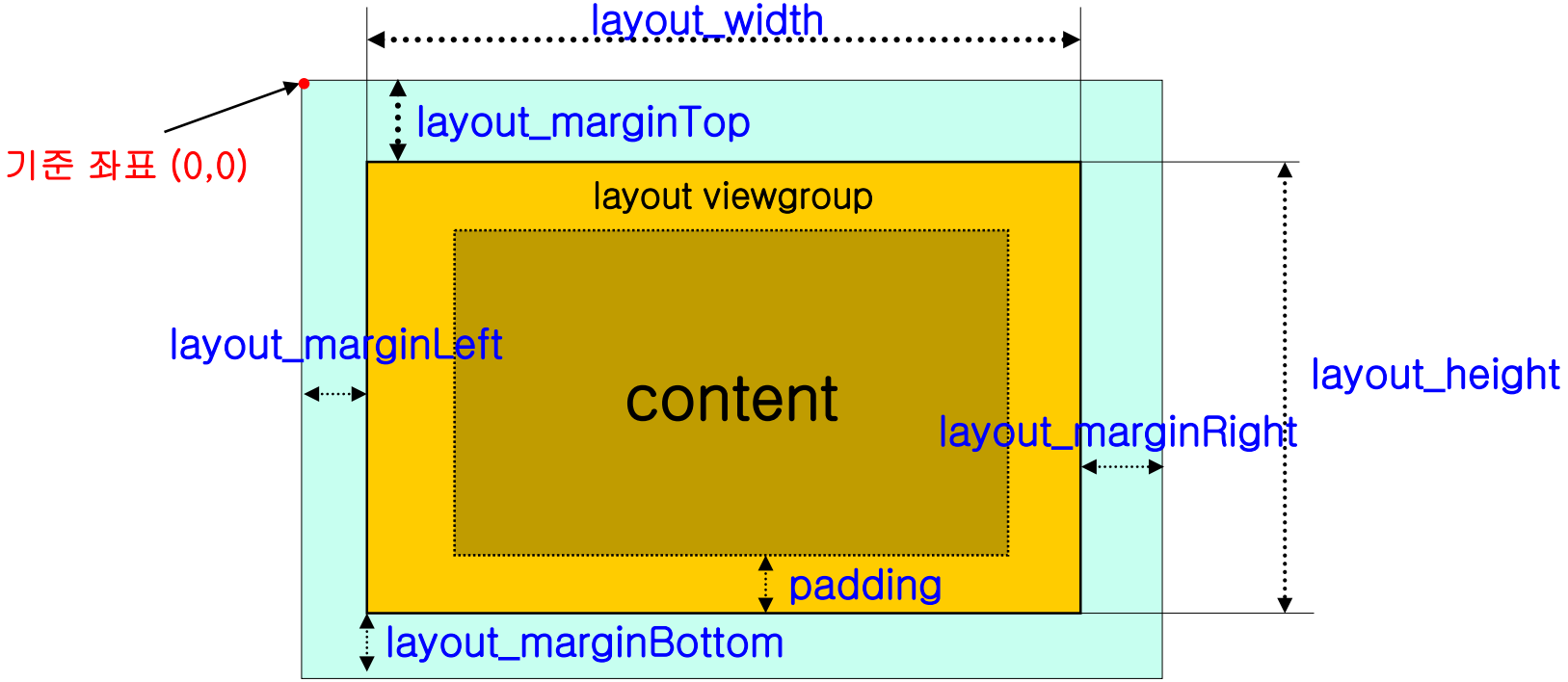
```
java.lang.Object
    └ android.view.View
        └ android.widget.ViewGroup
            └ android.widget.LinearLayout
                └ android.widget.TableLayout
            └ android.widget.RelativeLayout
            └ android.widget.FrameLayout
            └ android.widget.GridLayout
```

Layout 속성

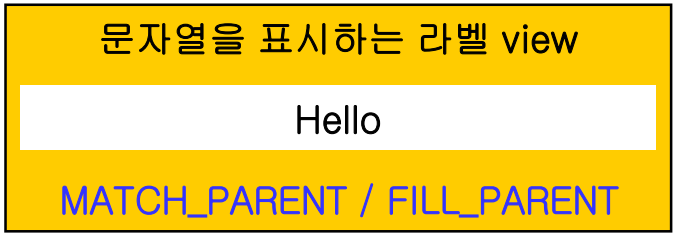


Layout 속성

- Layout의 좌표와 용어



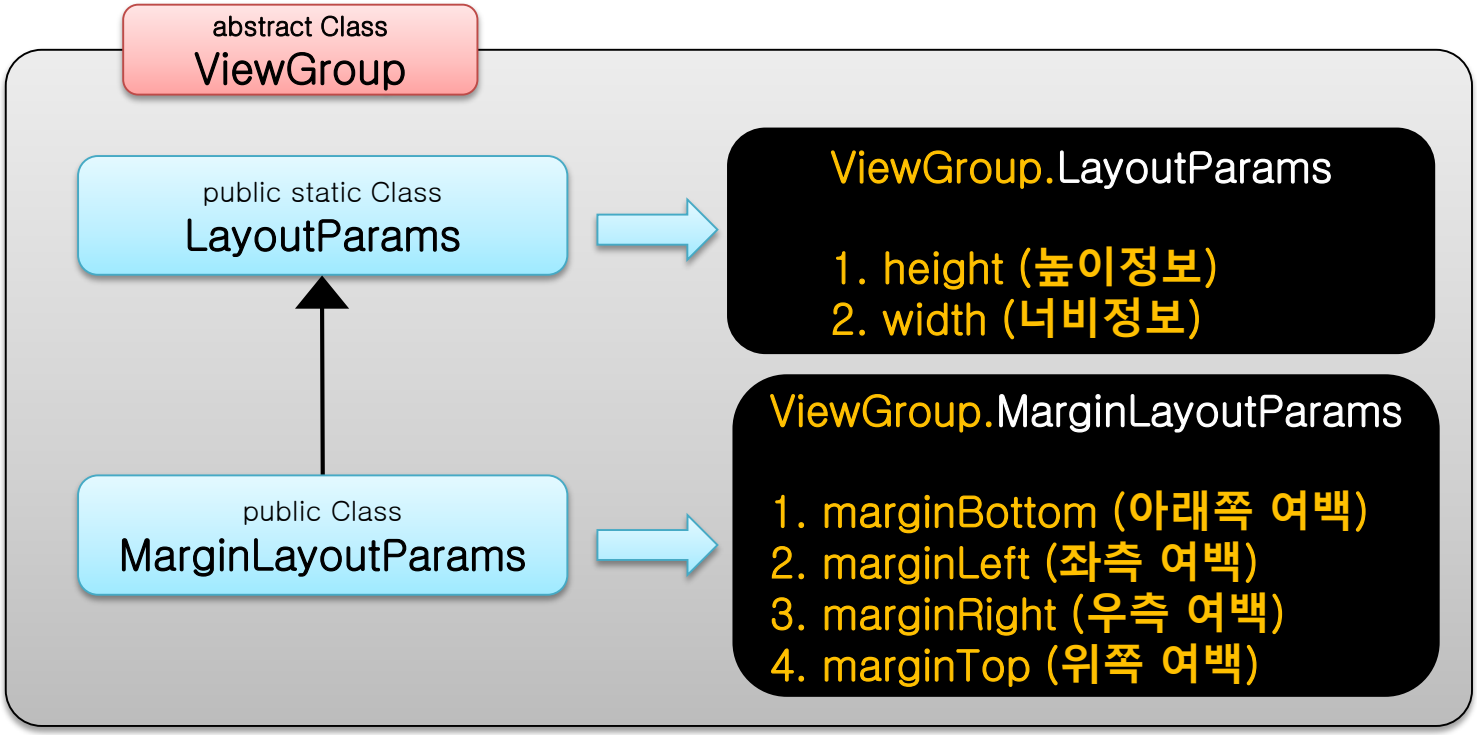
컨텐츠를 표시하는 데 충분한 크기로



부모 객체와의 패딩(여백)을 제외한 나머지 공간을 차지

Layout 속성

- 최상위 ViewGroup의 Layout 속성(Params) 정보



- LayoutParams 정보와 View 자체 속성에 대한 정보 구분은 "layout_" Prefix로 구분할 수 있음

Layout 속성

- “layout_” 속성
  - View(또는 ViewGroup)가 자신의 배치 관련 설정을 부모 Layout에게 요청하는 역할을 한다는 것
  - View에 사용되는 대부분의 속성들이 View 자체의 형태를 위한 속성이라면, "layout_"으로 시작하는 속성은 부모 Layout 내부에서 View가 가지는 크기, 여백, 위치 등을 설정하기 위한 속성인 것
  - "layout_"으로 시작하는 속성들을 사용한다고 해서, 그 속성 값이 View의 배치에 그대로 적용되는 것은 아님
  - Layout 내에 있는 다른 View Widget들의 속성 값 및 관계가 먼저 고려된 다음에 적용

Layout 속성

- Layout의 크기 ← 모든 뷰의 필수 속성
  - Layout이 화면에서 얼마만큼의 크기를 차지하게 할 것인가를 정할 수 있음
  - android:layout_width 속성
    - 이 속성은 Layout의 가로 크기를 지정
  - android:layout_height 속성
    - 이 속성은 Layout의 세로 크기를 지정
  - 가로 크기와 세로 크기에 대해서 숫자, 비율, Layout이 가지는 자식들의 크기값으로 지정

속성	부모 크기 만큼	가질 데이터 크기 만큼	pixel 또는 dp(dip)
android:layout_width	match_parent	wrap_content	480px 또는 320dp
android:layout_height	match_parent	wrap_content	800px 또는 533dp

Layout 속성

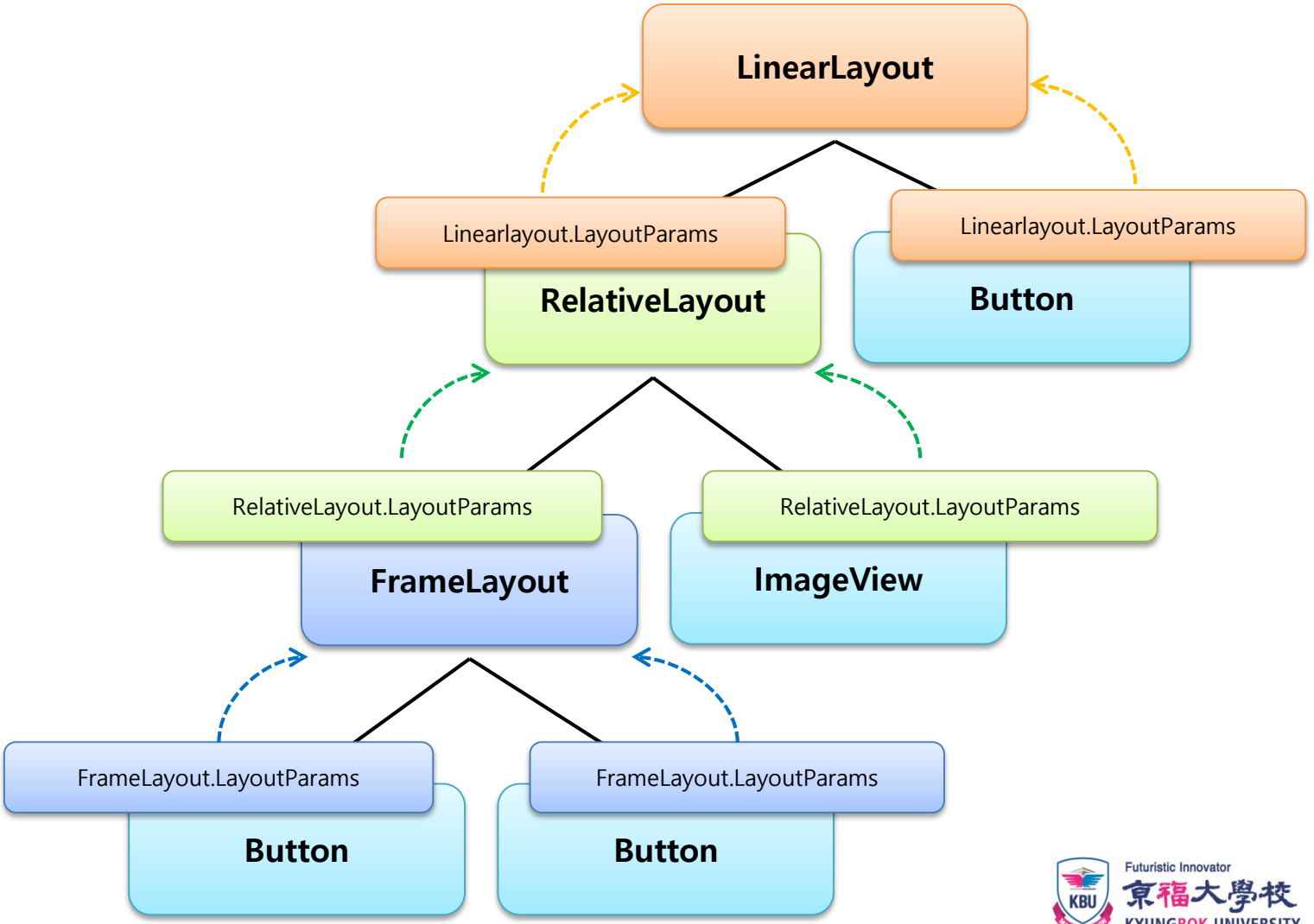
- Layout의 크기
  - 부모 크기 만큼
    (부모의 전체 크기 중에서, 내가 지금 작성한 Layout보다 먼저 작성된 Layout을 제외하고 남은 크기)
  - 현재 Android SDK 버전이 올라가면서 SDK 2.2부터는 'match_parent'라는 값이 나오고 있음
    - 이 값은 기존의 'fill_parent'와 같은 값
  - 부모 크기처럼 지정할 때에는 'match_parent'라고 설정해 줌

Layout 속성

- orientation
  - Layout 안에 배치할 Widget의 수직(Vertical) 또는 수평(Horizontal) 방향을 설정
- gravity
  - Layout 안에 배치할 Widget의 정렬 방향을 좌측(Left), 우측(Right), 중앙(Center)으로 설정
- padding
  - Layout 안에 배치할 Widget의 여백을 설정
- layout_weight
  - Layout이 전체 화면에서 차지하는 공간의 가중 값을 설정
  - 여러 개의 Layout이 중복될 때 주로 사용
- baselineAligned
  - Layout 안에 배치할 Widget들을 보기 좋게 정렬

Layout 속성

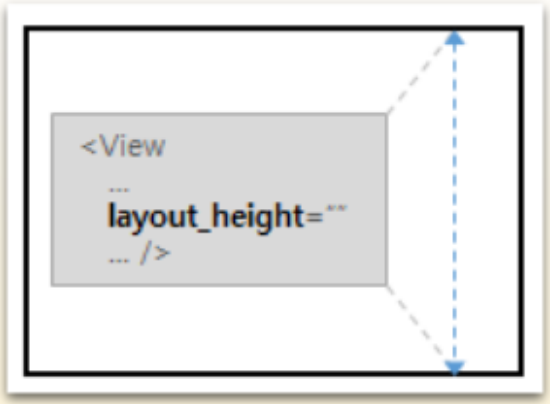
- View가 가지는 다양한 Layout 속성(Params) 정보

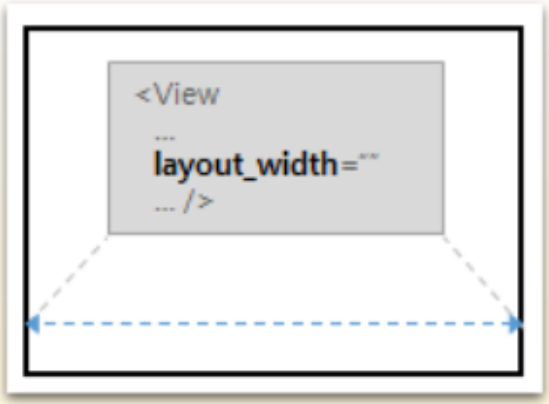


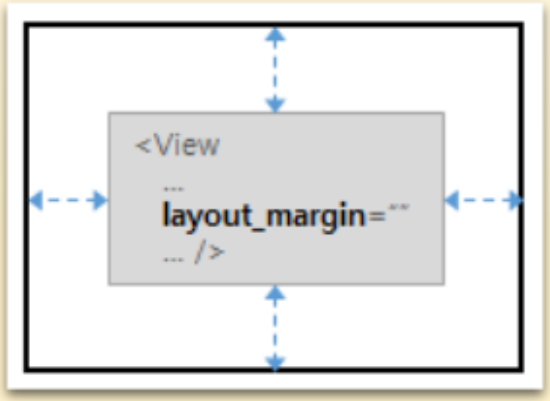
Layout 속성

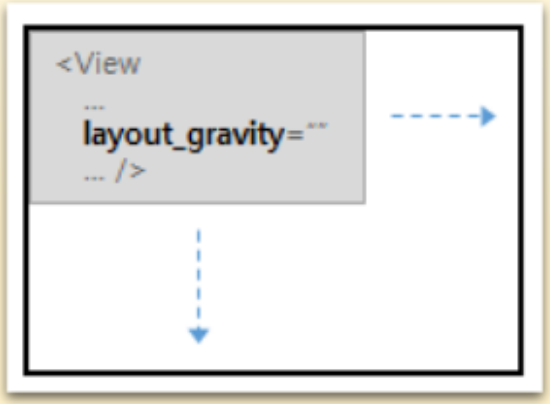
- layout_margin, padding
  - Layout에 자식 View Widget들이 표시될 때, Layout과 View Widget 사이 또는 인접한 두개의 View Widget 사이에 여백을 위한 공간이 존재하지 않으면, App이 보여주고자 하는 내용의 가독성과 사용 편의성이 현저히 떨어지게 됨

Layout 속성



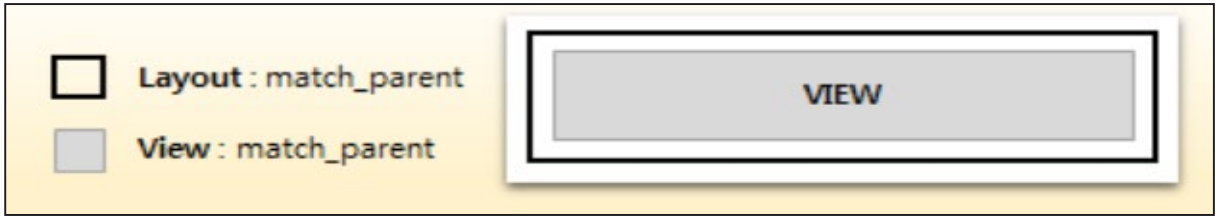


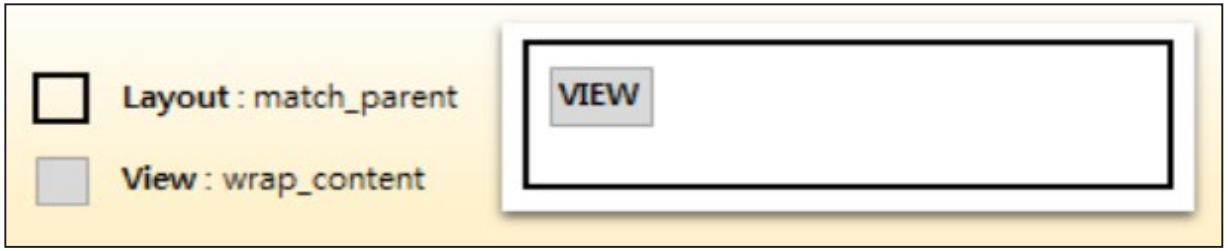


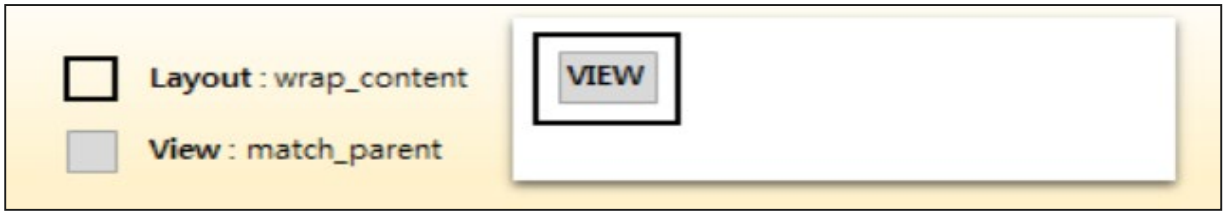


Layout 속성

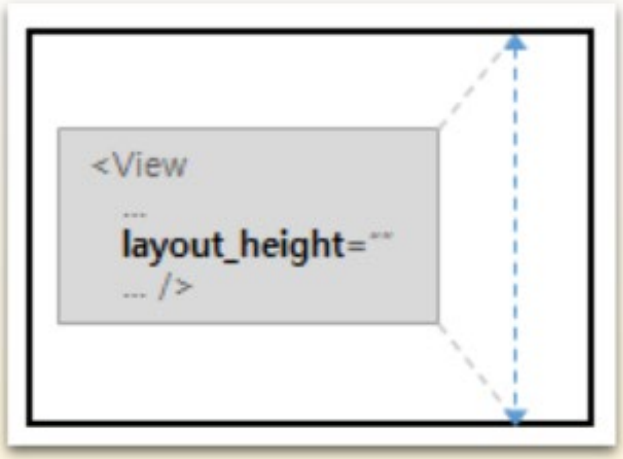
- Layout(검은 실선 영역)과 자식 View Widget과의 관계

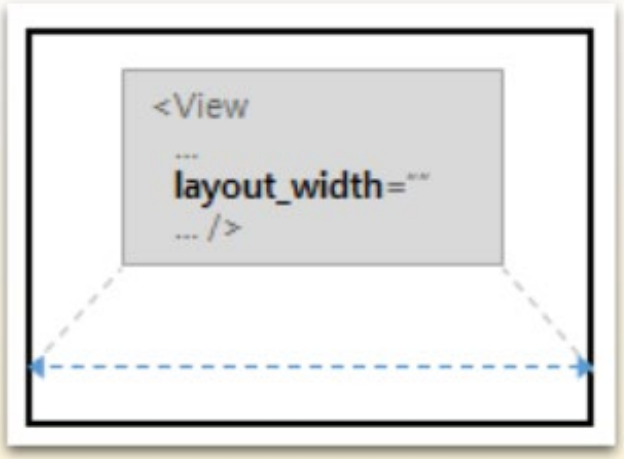


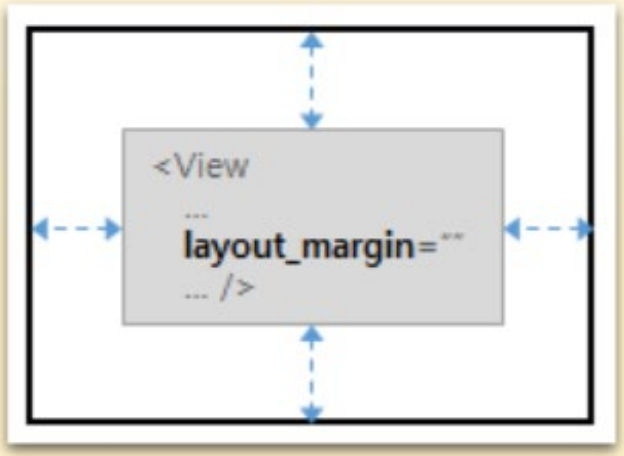


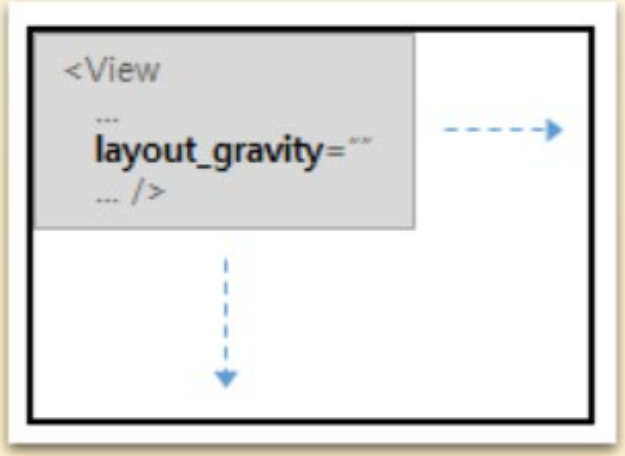


Layout 속성


<View
...
layout_height=""
... />

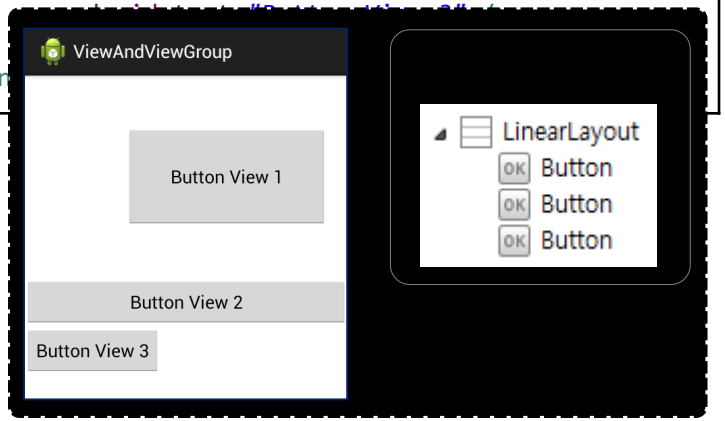

<View
...
layout_width=""
... />

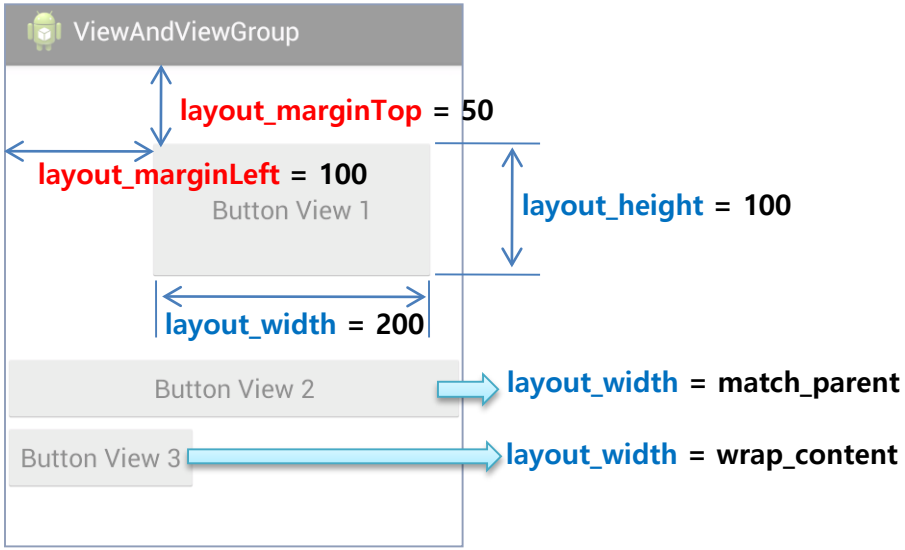

<View
...
layout_margin=""
... />

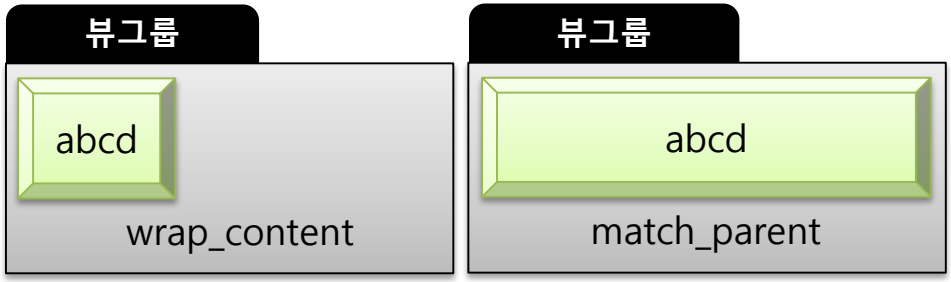

<View
...
layout_gravity=""
... />

Layout 속성

```
res/layout/activity_main.xml
<LinearLayout xmlns:android="…"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <Button
      android:layout_width="200dp"
      android:layout_height="100dp"
      android:layout_marginTop="50dp"
      android:layout_marginLeft="100dp"
      android:layout_marginBottom="50dp"
      android:text="Button View 1" />

    <Button
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="Button View 2" />

    <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
</Lin
```

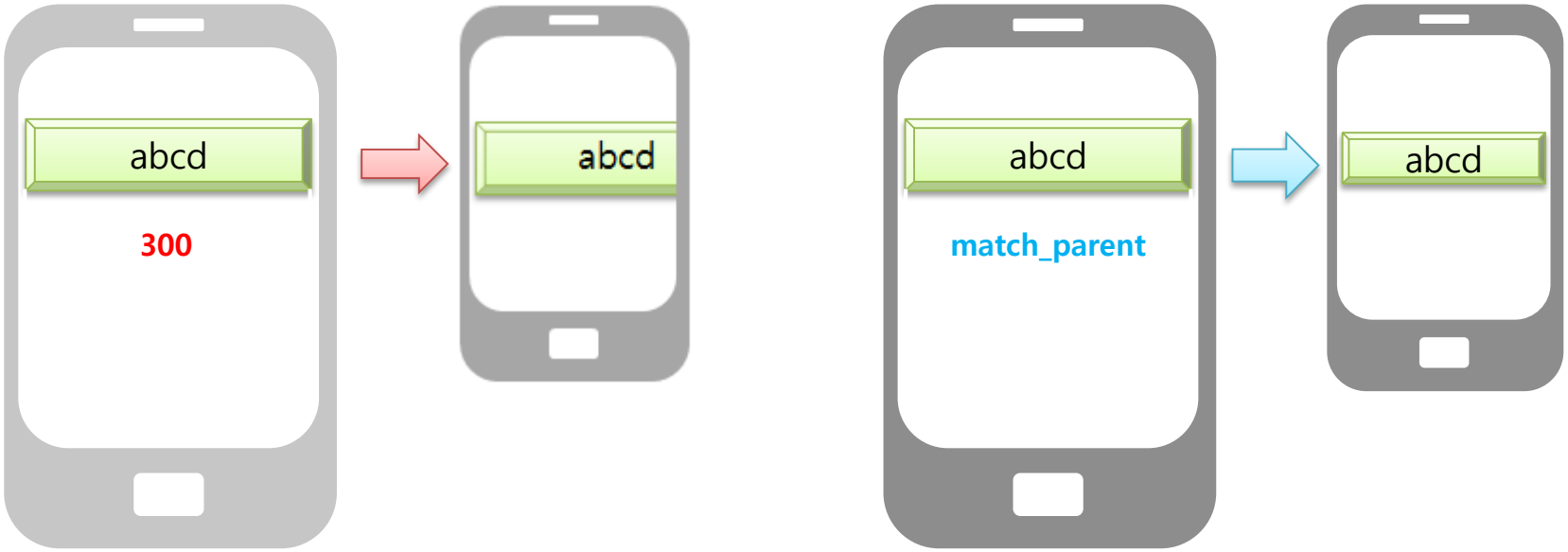






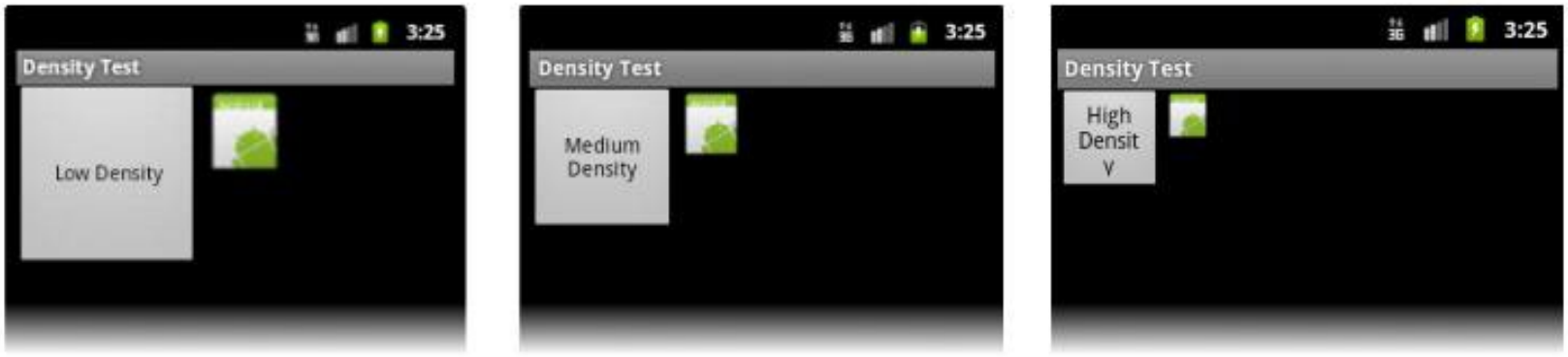
Layout 속성

- match_parent는 부모 WiewGroup에 크기를 맞추기 때문에 화면 크기가 다른 단말에서도 유연하게 Layout을 유지할 수 있음



Layout 속성

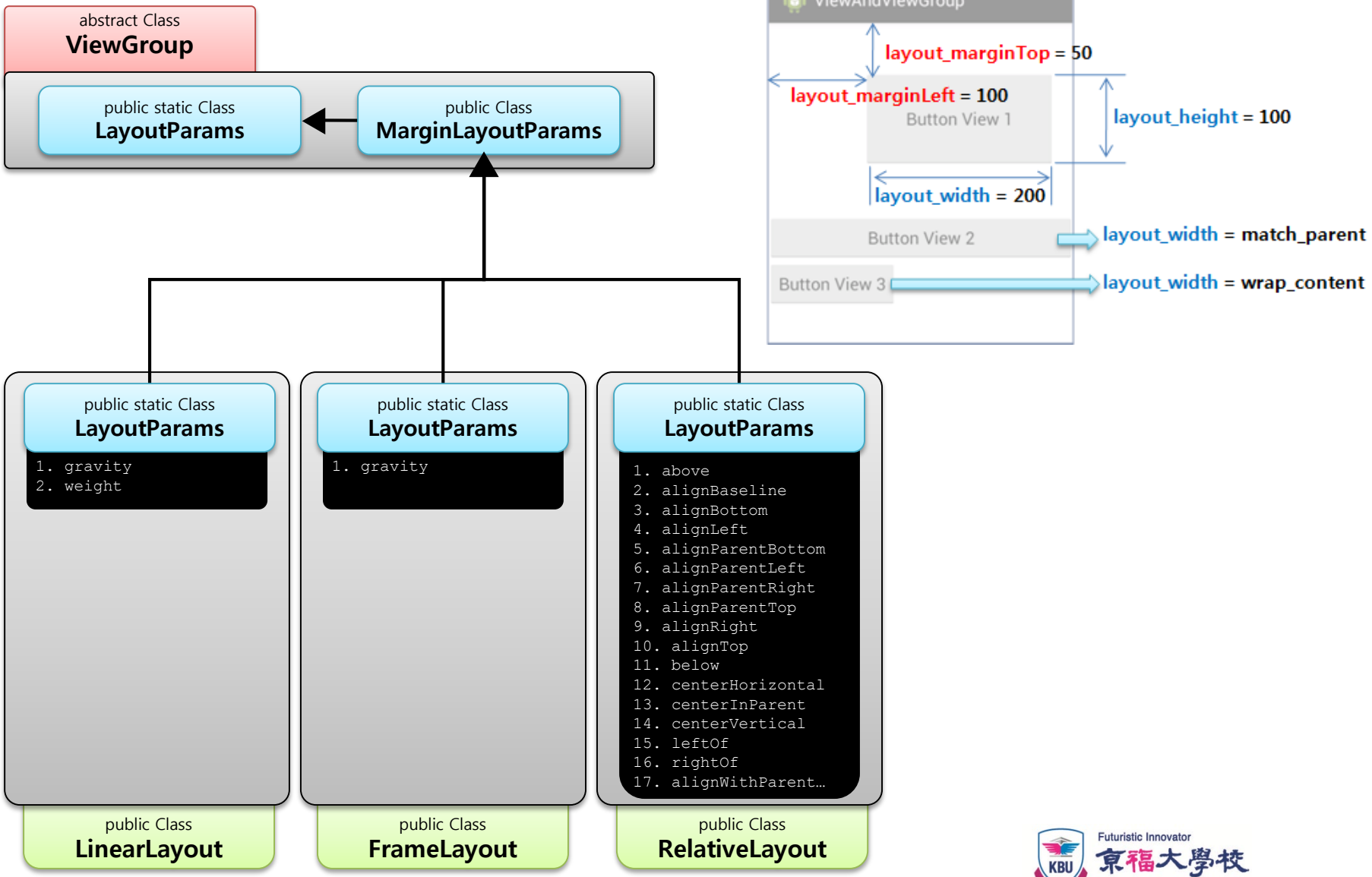
- DIP(Density independent pixel) 사용
  - PX 사용



  - DIP 사용

출처 : http://developer.android.com/guide/practices/screens_support.html

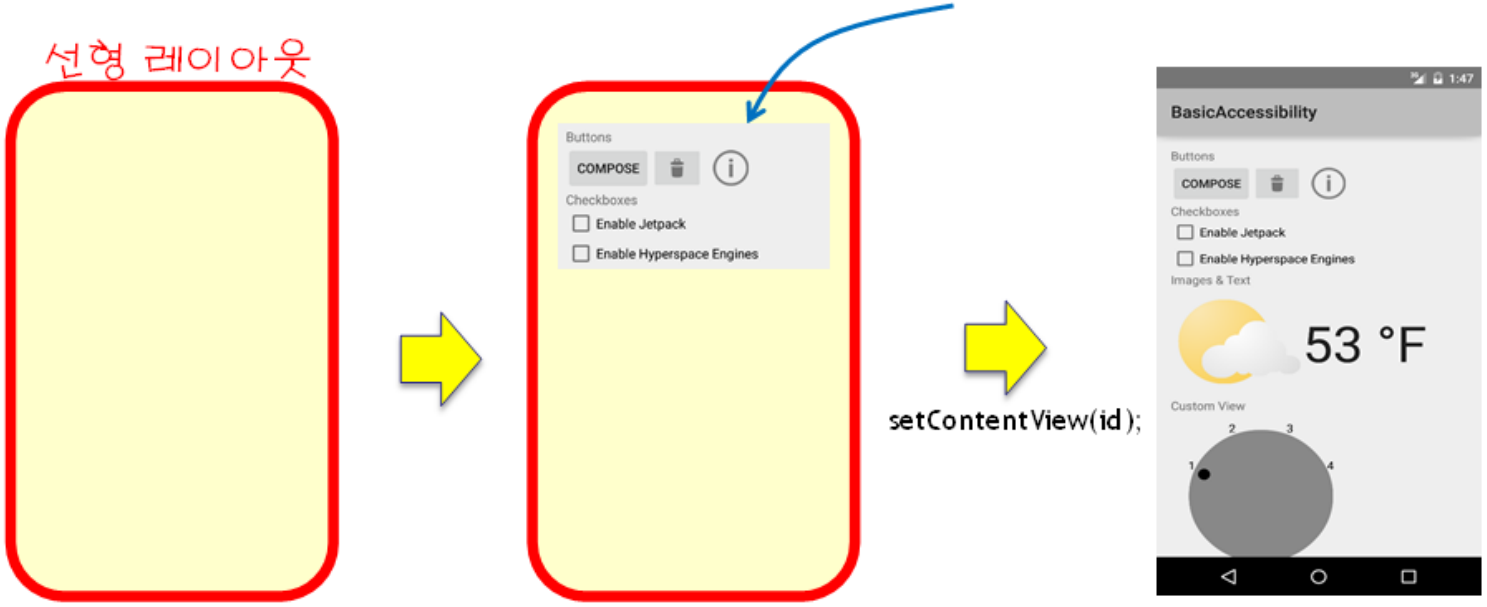
Layout 속성


abstract Class
ViewGroup
public static Class
LayoutParams
public Class
MarginLayoutParams
public static Class
LayoutParams
1. gravity
2. weight
public Class
LinearLayout
public static Class
LayoutParams
1. gravity
public Class
FrameLayout
public static Class
LayoutParams
1. above
2. alignBaseline
3. alignBottom
4. alignLeft
5. alignParentBottom
6. alignParentLeft
7. alignParentRight
8. alignParentTop
9. alignRight
10. alignTop
11. below
12. centerHorizontal
13. centerInParent
14. centerVertical
15. leftOf
16. rightOf
17. alignWithParent…
public Class
RelativeLayout

ViewAndViewGroup
layout_marginTop = 50
layout_marginLeft = 100
Button View 1
layout_height = 100
layout_width = 200
Button View 2
layout_width = match_parent
Button View 3
layout_width = wrap_content

Layout

- 작성 절차
  1. View Group을 생성
  2. 필요한 View를 추가
  3. Activity 화면으로 설정



① 뷰그룹을 생성한다.

② 필요한 뷰들을 추가한다.

③액티비티의 화면으로 설정한다.

Layout

- 작성 절차
  - UI에 사용되는 Widget(View)들을 이해
    - 어떤 Widget이 어디에 적합하고, 어떤 Widget이 어디에 쓰이면 좋을지 판단할 수 있는 것
  - 각 Widget의 속성을 이해
    - 각 Widget별로 속성들이 다름
    - 공통된 속성들도 있지만, 다른 것들도 많기 때문에 이것들을 알아야 함
    - Widget에 어떤 속성을 적용했을 때, Widget이 화면에 어떻게 보이는 지를 이해하고, 내 마음대로 Widget을 제어 할 수 있어야 함

Layout

- 작성 절차
  - Widget과 Widget의 관계를 이해
    - Widget들이 배치되어 있을 때, 어떤 Widget이 다른 Widget에 영향을 미칠 수 있음
    - Widget의 관계에 따라 서로 어떤 영향을 끼칠 수 있는지, 또 Widget의 관계(부모, 자식, 형제)에 따라 어떻게 보여질 수 있고 어떻게 배치해야 하는지를 알아야 함

Layout

- 작성 절차
  - 화면 설계
    - 일반적으로 화면에 보이는 것을 그대로 나타내기 위해선 화면을 보면서 화면에 대한 설계를 해야 함
    - Android의 Layout은 Tree 구조로 되어있기 때문에, 설계 없이는 좋은 Layout을 만들기 어려움
    - 양질의 Layout을 만들기 위해서는 하나로 보이는 화면을 분할하고 영역을 나누고, Widget들을 화면과 Mapping시켜서 Tree 구조로 설계를 해야 함
    - 예) Login 화면을 만든다면, 단순히 EditText 2개와 Button을 배치하면 끝나는 것이 아니라, 2개의 EditText가 위아래로 있고, 이 EditText 오른쪽에 로그인 Button을 놓고 싶다면, 3개의 Widget(EditText 2개, Button 1개)으로 끝나지 않는다는 것

Layout

- 작성 절차
  - 예쁘게 꾸밈
    - 위의 4가지를 숙지했다면, 이제 만들기만 하면 됨
    - Widget들을 이해하고, 화면을 구성할 수 있게 되었다면, 웬만한 화면을 XML로 구성하는 것은 쉬움
    - Mobile Programming은 사용자에게 보여지는 부분이 70%이상 임

Layout

- UI를 작성하는 3가지 방법
  - XML 기반 방법(선언적 Design 방법, 기본적인 방법)
    - XML를 이용해서 모든 UI를 선언
    - Static Layout(정적 레이아웃)
    - 선언 후에는 Runtime에 각 객체를 조정할 수 있음
    - XML Layout은 독립된 File이기 때문에 화면이 바뀔 때마다 계속해서 변경
  - JAVA 기반 방법(절차적 Design 방법)
    - 실행 시간에 JAVA Code로 UI를 생성
    - Dynamic Layout(동적 레이아웃)
    - JAVA Coding을 통해 Layout과 Button 등의 객체를 구성
    - JAVA Code로 직접 작성해도 XML로 Layout을 지정하는 모든 기능을 완벽하게 처리할 수 있음

Layout

- UI를 작성하는 3가지 방법
  - Hybrid 방법
    - 최초로 보이는 UI는 XML로 UI 선언
    - 추후는 JAVA Code로 UI 속성 수정
  - 선언적 방식의 장점
    - Application의 외형과 동작을 구분할 수 있음
    - Code와 Design의 분리
    - JAVA Code에 비해 짧고 이해하기 쉬움
    - JAVA로 작성된 기능에 Error가 생기는 문제를 예방
    - 구성된 화면을 쉽게 가져와 사용할 수 있음

Layout

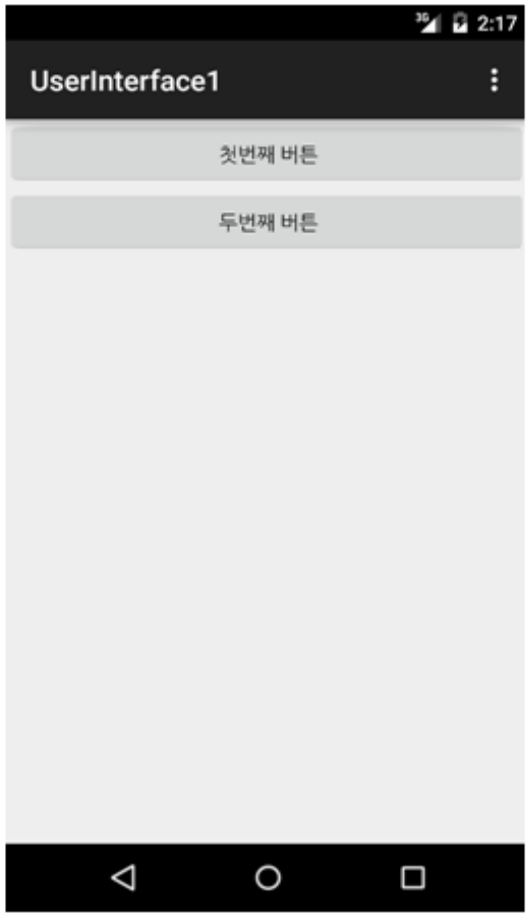
XML 기반 방법

① XML로 사용자 인터페이스 기술

```
...
<Button
android:text="첫번째 버튼"
android:id="@+id/button1"
android:layout_width="match_parent"
android:layout_height="wrap_content">
</Button>
...
```

② 코드로 사용자 인터페이스 작성

```
...
Button b1 = new Button(this);
b1.setText("첫번째 버튼");
container.addView(b1);
...
```



Layout

- XML 기반 방법
  - 각 Widget이 서로 연결되고 포함되는 관계를 XML 형태로 정의
  - Android는 XML Layout을 Resource File로 인식함
  - XML Layout File은 Project 내부의 res/layout Directory 아래 보관
  - 각 XML Layout File에 정의된 Element Tree 구조는 실제 화면에 나타나는 Widget과 Container의 구조를 그대로 나타냄
  - XML Element에 지정된 속성은 Widget 속성에 연결돼 Widget이 표현되는 방법이나 Container가 동작하는 방법 등을 정의
  - 예) Button Element에 android:textStyle="bold" 라고 지정되 있으면 해당 Button이 화면에 표시될 때 굵은 글꼴을 사용한다는 의미

Layout

- XML로 Layout을 지정하는 이유
  - JAVA Code와의 독립성
    - JAVA Code와 XML Layout File은 서로가 변경되더라도 서로에게 영향을 끼치지 않음
  - GUI 화면 Design Program에서 화면을 좀 더 쉽게 생성하고 관리하겠다는 목표

Layout

- XML Layout File 구조

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

  - XML File의 Root Element는 Android XML namespace를 지정해야 함

```
xmlns:android="http://schemas.android.com/apk/res/android"
```

  - android:text 속성
    - Button이 처음 생성 됐을 때 화면에 표시할 Text를 의미

Layout

- XML Layout File 구조
  - android:layout_width / android:layout_height 속성
    - 자신을 포함하는 부모(parent) Element의 폭과 너비를 그대로 사용
  - Element ID
    - JAVA Code에서 호출해 사용하려는 모든 Element들은 반드시 android:id 속성으로 id를 지정해야 함
    - 일반적으로 @+id/... 와 같은 Id를 지정
    - ...부분에는 중복되지 않는 유일한 문자열을 지정해야 함
    - 예) android:id="@+id/button”
    - Andropid에서는 @android:id/...와 같이 특별한 형태의 android:id값을 사용하기도 함

Layout

- JAVA Code와 연결하는 방법
  - 필요한 Widget과 Container를 구성해 main.xml 파일로 저장하고 res/layout Directory에 보관
  - Activity의 onCreate() 메소드에서 다음과 같은 Code 한 줄만 적어주면 작성한 Layout을 그대로 불러옴

```
setContetView(R.layout.main);
```

Layout

- JAVA Code와 연결하는 방법
  - Element 가운데 id를 지정한 항목 찾아오기
    - findViewById() 메소드를 사용
    - 메소드 인자로는 찾고자 하는 Widget의 숫자 ID를 넘겨야 함
    - Widget의 숫자 ID는 Android가 자동으로 생성해 R 클래스에 반영, R.id.something으로 사용(something 부분에 Widget 이름)
  - R.java
    - XML Layout File을 Android Build System이 분석해 JAVA Code에서 호출해 쓸 수 있도록 자동 생성한 JAVA Code File
    - 모든 Layout 정보는 R.layout 변수를 통해 접근, Layout File 이름으로(main.xml File의 Layout은 R.layout.main) 접근

Layout

- JAVA Code와 연결하는 방법
  - Layout을 XML로 분리한 Program은 대부분 JAVA로만 구성된 Program과 동일하지만,
  - Widget의 Instance를 생성하지 않고, XML Layout에 정의된 내용을 호출해서 사용한다는 점이 다름

Layout

- Hybrid 방법
  - 각 Widget이 서로 연결되고 포함되는 관계를 XML 형태로 정의
  - Android는 XML Layout을 Resource File로 인식하기 때문에 XML Layout File은 Android 내부의 [res]-[layout] Directory에 보관
  - 각 XML Layout File에 정의된 Element Tree 구조를 보면 실제 화면에 나타나는 Widget과 Container의 구조를 그대로 나타냄

Layout

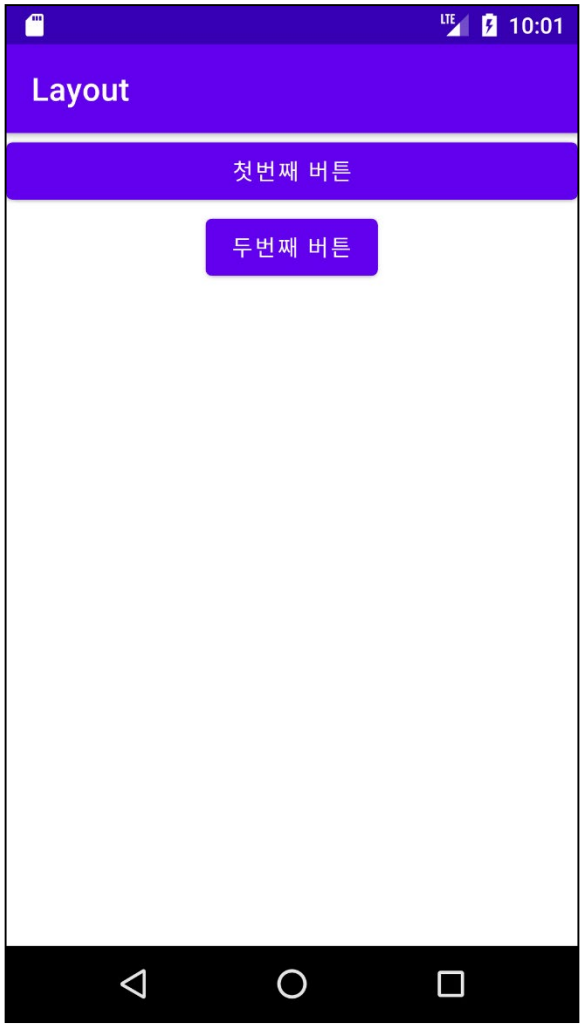
- Android에서 화면 : Source와 화면 구성이 분리되어 있음
  - JAVA Source 1개
  - XML Layout 1개

- 화면 전체 : Activity (setContentView()에서 Inflation)
  - Activity를 위한 JAVA 소스 1개 : MainActivity.java
  - Activity를 위한 XML Layout 1개 : activity_main.xml

- 부분 화면(수동으로 Inflation)
  - 부분 화면을 위한 JAVA Source 1개 또는 View
    (View가 1개의 Source File로 분리될 수 있음)
  - 부분 화면을 위한 XML Layout 1개 : single.xml

Layout 예제 1

- 다음과 같이 화면에 표시되도록 3가지 방법을 이용하여 만들어보자



Layout 예제 1

- UI를 작성하는 3가지 방법
  - XML로 UI 작성

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

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="첫번째 버튼"/>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="두번째 버튼" />
</LinearLayout>
```

선형 레이아웃 이라는 뷰 그룹을 생성

버튼 이라는 뷰를 생성

Layout 예제 1

- JAVA Code로 View를 생성하는 방법

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    LinearLayout layout = new LinearLayout(this);
    layout.setLayoutParams(new LinearLayout.LayoutParams(
            LinearLayout.LayoutParams.MATCH_PARENT,
            LinearLayout.LayoutParams.MATCH_PARENT
    ));
    layout.setOrientation(LinearLayout.VERTICAL);

    Button button1 = new Button(this);
    button1.setLayoutParams(new LinearLayout.LayoutParams(
            LinearLayout.LayoutParams.MATCH_PARENT,
            LinearLayout.LayoutParams.WRAP_CONTENT
    ));
    button1.setText("첫번째 버튼");
    layout.addView(button1);
```

선형 레이아웃 생성

버튼을 선형 레이아웃에 추가

Layout 예제 1

- JAVA Code로 View를 생성하는 방법

```
    Button button2 = new Button(this);
    LinearLayout.LayoutParams button2Params = new LinearLayout.LayoutParams(
            LinearLayout.LayoutParams.WRAP_CONTENT,
            LinearLayout.LayoutParams.WRAP_CONTENT
    );
    button2Params.gravity = android.view.Gravity.CENTER;
    button2.setLayoutParams(button2Params);
    button2.setText("두번째 버튼");
    layout.addView(button2);
    setContentView(layout);
}
```

Layout 예제 1

- XML과 JAVA Code를 동시에 사용하는 방법(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"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity10">

    <Button
        android:id="@+id/button1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center" />
</LinearLayout>
```

버튼에
식별자를 부여

Layout 예제 1

- XML과 JAVA Code를 동시에 사용하는 방법(JAVA Code)

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main10);

    Button button1 = findViewById(R.id.button1);
    Button button2 = findViewById(R.id.button2);

    button1.setText("첫번째 버튼");
    button2.setText("두번째 버튼");

    button2.setBackgroundColor(Color.BLUE);
    button1.setEnabled(false);
}
```

버튼을 연결함

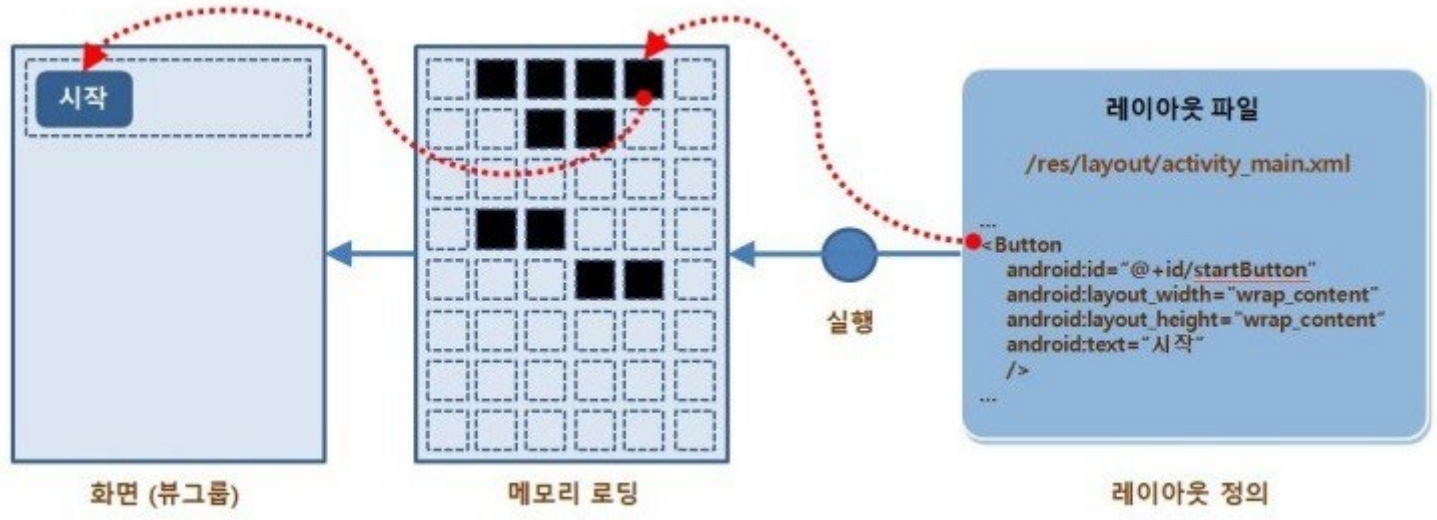
버튼의 속성을 변경

LayoutInflater

- XML의 Layout은 화면 배치만 정의할 뿐임
- 여기에 화면의 기능을 담당하는 JAVA Source File이 필요
  - 즉, 하나의 화면이 동작하기 위해서는 Main Layout XML File 하나와 JAVA Source File이 필요
- 이렇게 화면의 구성과 기능을 분리하는 이유는 화면만 따로 분리하면 이해도 쉽고 관리도 편리하기 때문임
- 처음 App을 만들었을 때 보이는 화면 역시 JAVA Source Code에서 화면을 설정한 것
- 이 부분은 setContentView() 메소드가 담당
- MainActivity 클래스는 AppCompatActivity 클래스를 상속
  - AppCompatActivity 클래스에는 화면에 필요한 기능들이 정의되어 있음
- App 실행했을 때 화면이 보인다는 것은 화면 배치 정보(XML 코드)가 Memory에 Loading되어 객체로 존재한다는 것
- 이 과정을 Layout Inflation이라고 함

LayoutInflater

- LayoutInflater란?



  - Inflater 단어의 뜻이 “부풀리다”라는 의미로써 LayoutInflater라는 단어에서도 유추가 가능
  - LayoutInflater는 XML에 미리 정의해둔 틀을 실제 Memory에 올려주는 역할
  - LayoutInflater는 XML에 정의된 Resource를 View 객체로 반환해주는 역할

LayoutInflater

- LayoutInflater란?
  - 보통 JAVA Code에서 View, ViewGroup을 사용하거나, Adapter의 getView() 또는 Dialog, Popup 구현 시 배경 화면이 될 Layout을 만들어 놓고 View의 형태로 반환 받아 Activity에서 실행 하게 됨
  - 우리가 보통 Activity를 만들면 onCreate() 메소드에 기본으로 추가되는 setContentView(R.layout.activity_main) 메소드와 같은 원리
  - 이 메소드 또한 activity_main.xml 파일을 View로 만들어서 Activity위에 보여줌
  - 사용자의 화면에 보여지는 것들은 Activity 위에 있는 View 임

LayoutInflater

- setContentView() 메소드의 역할

```
public void setContentView (int layoutResID)
public void setContentView (View view [, ViewGroup.LayoutParams params])
```

  - Parsing
    - setContentView() 메소드는 Layout XML의 내용을 Parsing하여 View들을 생성하고, View에 정의된 속성들을 설정
    - 생성된 View들을 상호관계에 맞춰 배치
    - setContentView() 메소드 내에 이러한 처리는 모두 LayoutInflater라는 클래스를 참조하고 있음
  - Layout XML의 처리 결과로 생성된 View들을 Content 영역에 추가

LayoutInflater

- Android에서 Layout XML File을 View 객체로 만들기 위해서는 LayoutInflater를 이용
- LayoutInflater 생성 방법
  - getSystemService()
    - LayoutInflater 객체는 System Service 객체로 제공되기 때문에 getSystemService() 메소드를 이용해 참조할 수 있음

```
LayoutInflater inflater = (LayoutInflater) getSystemService(
                                    Context.LAYOUT_INFLATER_SERVICE);
View view = inflater.inflate(R.layout.my_layout, parent, false);
```

  - 전체 화면 중에서 일부분만을 차지하는 화면 구성요소들을 XML Layout에서 Loading하여 보여줄 수 없을까?
    - LayoutInflater라는 클래스를 제공하며, 이 클래스는 System Service로 제공됨

LayoutInflater

- LayoutInflater 생성 방법
  - LayoutInflater.from() 메소드
    - LayoutInflater에서는 LayoutInflater를 쉽게 생성 할 수 있도록 static factory 메소드 LayoutInflater.from()을 제공 (내부적으로 getSystemService를 호출함)

```
LayoutInflater inflater = LayoutInflater.from(context);
View view = inflater.inflate(R.layout.my_layout, parent, false);
```

LayoutInflater

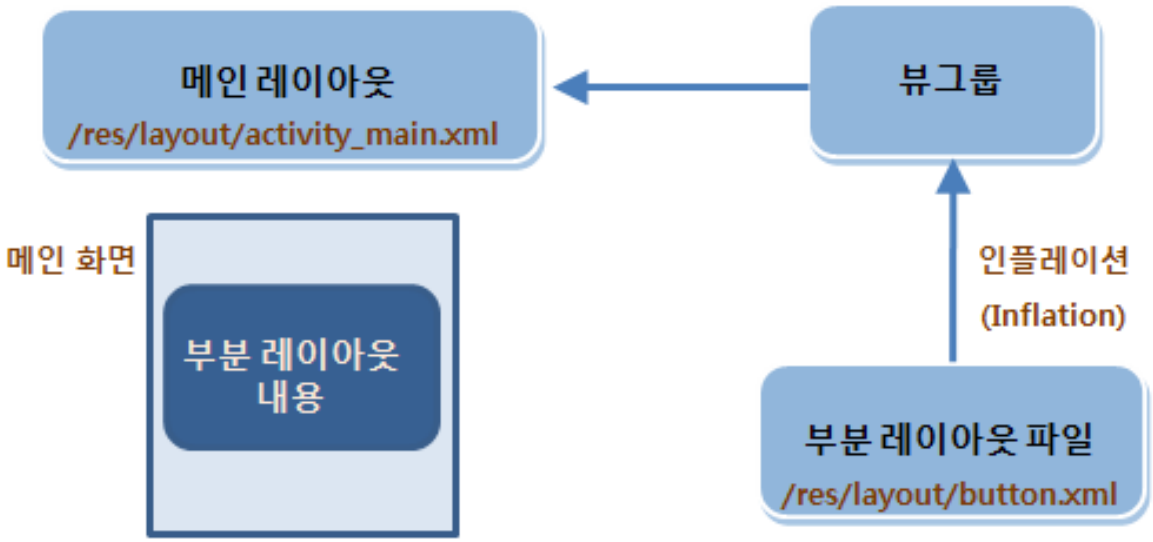
- LayoutInflater 생성 방법
  - Activity.getLayoutInflater()
    - Activity에서는 LayoutInflater를 쉽게 얻어올 수 있도록 getLayoutInflater() 메소드를 제공
    - Activity의 Window에 있는 getLayoutInflater()로 포워딩

```
LayoutInflater inflater = getLayoutInflater();
```

```
// android/app/Activity.java
public LayoutInflater getLayoutInflater() {
    return getWindow().getLayoutInflater();
}
```

LayoutInflater

- Layout Inflation의 개념도



[화면의 일부분을 XML 레이아웃 파일의 내용으로 적용하는 과정]

LayoutInflater 사용하기

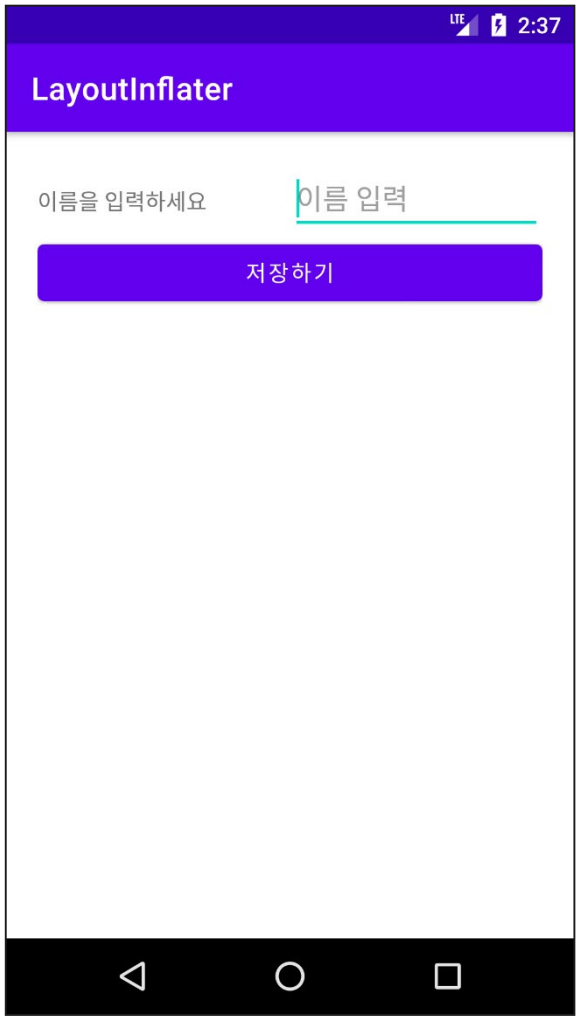
- View Factory 메소드

```
View view = View.inflate(context, R.layout.my_layout, parent);
```

  - View에서는 LayoutInflater의 inflate까지 한번에 실행하는 View.inflate() 메소드를 제공
  - 내부에서는 LayoutInflater.inflate를 수행함
  - 주의할 점은 parent를 지정한다면 자동으로 attach됨

```
public static View inflate(Context context, int resource,
                                                    ViewGroup root) {
    LayoutInflater factory = LayoutInflater.from(context);
    return factory.inflate(resource, root);
}
```

Layout 예제 2

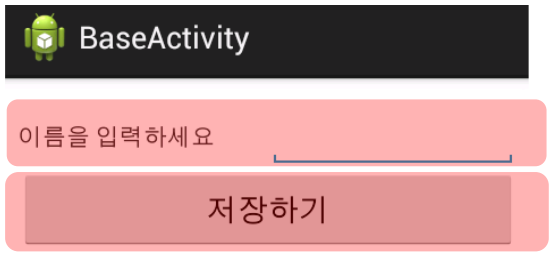


Layout 예제 2

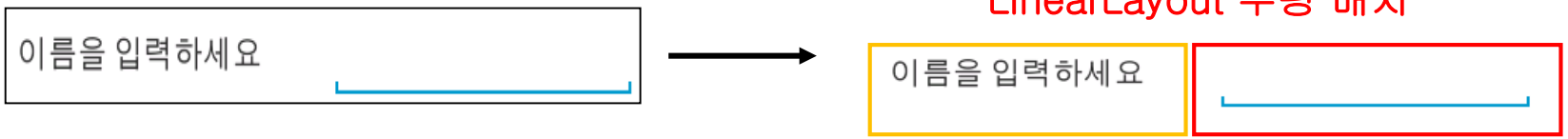
- 화면 설계
  - 일반적으로 화면에 보이는 것을 그대로 나타내기 위해선 화면을 보고 화면에 대한 설계를 해야 함
  - Android의 Layout은 Tree 구조로 되어있기 때문에, 설계 없이는 좋은 Layout을 만들기는 어려움
  - 양질의 Layout을 만들기 위해서는 보이는 화면을 하나로 보고 각 각을 분할하여 영역을 나누고, Widget들을 화면과 Mapping하여 Tree 구조로 설계를 해야 함
  - 예), 로그인 화면을 만든다고 했을 때, 단순히 EditText 2개와 Button을 배치하고 끝나는 것이 아니라, 두 개의 EditText가 위아래로 있고, 이 EditText 오른쪽에 로그인 Button을 놓고 싶다면, 3개의 Widget(EditText 2개, Button 1개)으로 끝나지 않는다는 것

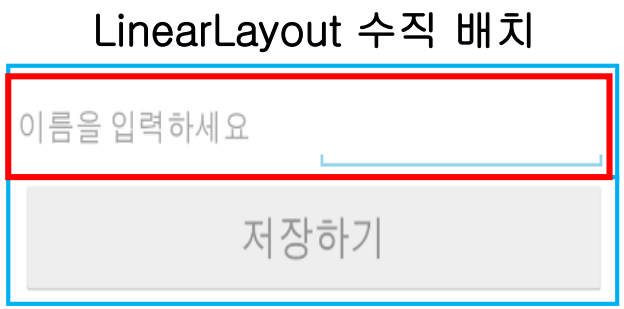
Layout 예제 2

- Layout 만들기
  - 화면을 설계하고, Tree 구조로 구분



아래의 Widget이 수직으로 구성
TextView, EditText가 수평으로 배치
Button 배치





Layout 예제 2

- Layout XML File 작성

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

Layout 예제 2

- Layout XML File 작성

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content“
    android:orientation=“horizontal" >    // default 값이므로 생략 가능
    <TextView
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="이름을 입력하세요 " />

    <EditText
        android:id="@+id/editText"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:hint="이름 입력" />
</LinearLayout>
```

Layout 예제 2

- Layout XML File 작성

```
    <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="저장하기" />
</LinearLayout>
```

Layout 예제 2

- JAVA Code로 Content 영역 만들기

```
public class MainActivity9 extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        LinearLayout  rootLayout = new LinearLayout(this);
        rootLayout.setOrientation(LinearLayout.VERTICAL);
        LinearLayout.LayoutParams root = new LinearLayout.LayoutParams(
                        LinearLayout.LayoutParams.MATCH_PARENT,
                        LinearLayout.LayoutParams.MATCH_PARENT);
        root.setMargins(20, 20, 20, 20);
```

Layout 예제 2

- JAVA Code로 Content 영역 만들기

```
LinearLayout inputLayout = new LinearLayout(this);
inputLayout.setOrientation( LinearLayout.HORIZONTAL );
LinearLayout.LayoutParams nameInput =
        new LinearLayout.LayoutParams(
                LinearLayout.LayoutParams.MATCH_PARENT,
                LinearLayout.LayoutParams.WRAP_CONTENT);
TextView textView = new TextView(this);
textView.setText( "이름을 입력하세요" );
LinearLayout.LayoutParams nameText = new LinearLayout.LayoutParams(
        0, LinearLayout.LayoutParams.WRAP_CONTENT);
nameText.weight = 1;
EditText editText = new EditText(this);
editText.setHint("이름 입력");
LinearLayout.LayoutParams nameEdit = new LinearLayout.LayoutParams(
        0, LinearLayout.LayoutParams.WRAP_CONTENT);
nameEdit.weight = 1;

inputLayout.addView(textView, nameText);
inputLayout.addView(editText, nameEdit);
```

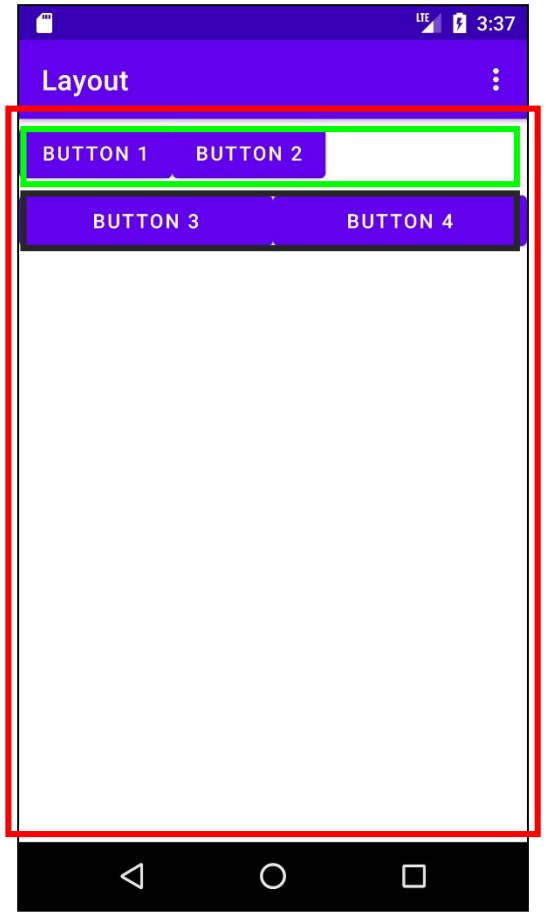
Layout 예제 2

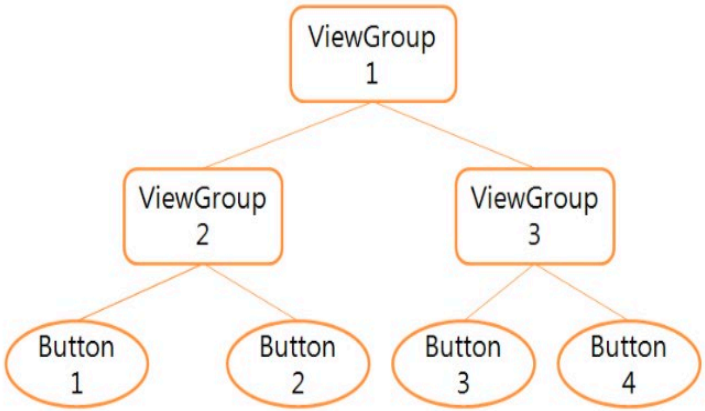
- JAVA Code로 Content 영역 만들기

```
        Button button = new Button(this);
        button.setText("저장하기");
        LinearLayout.LayoutParams saveButton = new LinearLayout.LayoutParams(
                        LinearLayout.LayoutParams.MATCH_PARENT,
                        LinearLayout.LayoutParams.WRAP_CONTENT);

        rootLayout.addView(inputLayout, nameInput);
        rootLayout.addView(button, saveButton);
        setContentView(rootLayout);
    }
}
```

Layout 예제 3

- View Architecture
  - View 클래스의 객체들을 계층 구조로 묶어서 사용





Layout 예제 3

- View Architecture
  - XML을 이용해서 계층 구조 표현(Sub Tag로 표현)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity10">
```

Layout 예제 3

- View Architecture
  - XML을 이용해서 계층 구조 표현(Sub Tag로 표현)

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 1" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 2" />
</LinearLayout>
```

Layout 예제 3

- View Architecture
  - XML을 이용해서 계층 구조 표현(Sub Tag로 표현)

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <Button
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Button 3" />

    <Button
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Button 4" />
</LinearLayout>
```

Layout 예제 3

- MainActivity.JAVA

```
public class MainActivity6 extends AppCompatActivity {
    int index = 1;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
    @Override
    protected void onStart() {
        super.onStart();
        if (index == 1)
            setContentView(R.layout.activity_main6);
        else if (index == 2){
            LinearLayout rootLayout = new LinearLayout(this);
            rootLayout.setLayoutParams(new LinearLayout.LayoutParams(
                    ViewGroup.LayoutParams.MATCH_PARENT,
                    ViewGroup.LayoutParams.MATCH_PARENT));
            rootLayout.setOrientation(LinearLayout.VERTICAL);
            addView(rootLayout);
            setContentView(rootLayout);
```

Layout 예제 3

- MainActivity.JAVA

```
    } else {
        View view = View.inflate(MainActivity6.this, R.layout.activity_addview, null);
        LinearLayout rootLayout = view.findViewById(R.id.layout);
        rootLayout.setBackgroundColor(Color.GREEN);
        addView(rootLayout);
        setContentView(rootLayout);
        setContentView(view);
    }
}
```

Layout 예제 3

- MainActivity.JAVA

```
private void addView(LinearLayout rootLayout) {
    LinearLayout nestedLayout1 = new LinearLayout(this);
    nestedLayout1.setLayoutParams(new LinearLayout.LayoutParams(
            ViewGroup.LayoutParams.MATCH_PARENT,
            ViewGroup.LayoutParams.WRAP_CONTENT));

    Button button1 = new Button(this);
    button1.setLayoutParams(new LinearLayout.LayoutParams(
            ViewGroup.LayoutParams.WRAP_CONTENT,
            ViewGroup.LayoutParams.WRAP_CONTENT));
    button1.setText("Button 1");
    nestedLayout1.addView(button1);
    Button button2 = new Button(this);
    button2.setLayoutParams(new LinearLayout.LayoutParams(
            ViewGroup.LayoutParams.WRAP_CONTENT,
            ViewGroup.LayoutParams.WRAP_CONTENT));
    button2.setText("Button 2");
    nestedLayout1.addView(button2);
    rootLayout.addView(nestedLayout1);
```

Layout 예제 3

- MainActivity.JAVA

```
    LinearLayout nestedLayout2 = new LinearLayout(this);
    nestedLayout2.setLayoutParams(new LinearLayout.LayoutParams(
            ViewGroup.LayoutParams.MATCH_PARENT,
            ViewGroup.LayoutParams.WRAP_CONTENT));
    Button button3 = new Button(this);
    LinearLayout.LayoutParams params3 = new LinearLayout.LayoutParams(
            0, ViewGroup.LayoutParams.WRAP_CONTENT, 1.0f);
    button3.setLayoutParams(params3);
    button3.setText("Button 3");
    nestedLayout2.addView(button3);

    Button button4 = new Button(this);
    LinearLayout.LayoutParams params4 = new LinearLayout.LayoutParams(
            0, ViewGroup.LayoutParams.WRAP_CONTENT, 1.0f);
    button4.setLayoutParams(params4);
    button4.setText("Button 4");
    nestedLayout2.addView(button4);
    rootLayout.addView(nestedLayout2);
}
```

Layout 예제 3

- MainActivity.JAVA

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.menu, menu);
    return true;
}
```

Layout 예제 3

- MainActivity.JAVA

```
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.item1:
                index = 1;
                break;
            case R.id.item2:
                index = 2;
                break;
            case R.id.item3:
                index = 3;
        }
        item.setChecked(true);
        onStart();
        return true;
    }
}
```

Layout 예제 3

- Android에서 Layout XML File을 View 객체로 만들기 위해서는 LayoutInflater를 이용해야 함

```
LayoutInflater inflater = (LayoutInflater) getSystemService(
                                        Context.LAYOUT_INFLATER_SERVICE);
View view = inflater.inflate(R.layout.my_layout, parent, false);
```

```
View view = getLayoutInflater().
                    inflate(R.layout.my_layout, parent, false);
```

  - 이렇게 간단하게 사용 할 수 있지만 LayoutInflater를 생성하는 법과 XML Layout을 inflate할 때 알아야 할 것과 주의해야 할 것을 자세하게 알아 보자

Layout 예제 3

- LayoutInflater 생성하기
  - getSystemService()

```
LayoutInflater inflater = (LayoutInflater) .getSystemService(
                                Context.LAYOUT_INFLATER_SERVICE);
View view = inflater.inflate(R.layout.my_layout, parent, false);
```

  - Activity의 getLayoutInflater() 사용
    - Activity에서는 LayoutInflater를 쉽게 얻어올 수 있도록 getLayoutInflater() 메소드를 제공
    - Activity의 Window에는 getLayoutInflater()로 포워딩 해줌

```
LayoutInflater inflater = getLayoutInflater();

public LayoutInflater getLayoutInflater() {
        return getWindow().getLayoutInflater();
}
```

Layout 예제 3

- LayoutInflater 생성하기
  - LayoutInflater Factory 메소드
    - LayoutInflater에서는 LayoutInflater를 쉽게 생성 할 수 있도록 static factory 메소드 LayoutInflater.from()을 제공
    - 내부에서는 getSystemService를 호출

```
LayoutInflater inflater = LayoutInflater.from(context);
View view = inflater.inflate(R.layout.my_layout, parent, false);
```

Inflation 예제 1

- LayoutInflater 생성하기
  - View Factory 메소드

```
View view = View.inflate(context, R.layout.my_layout, parent)
```

  - View에서는 LayoutInflater의 inflate까지 한번에 실행하는 View.inflate()를 제공
  - 내부에서는 LayoutInflater.inflate()를 수행
  - 주의할 점은 parent를 지정한다면 자동으로 attach 됨

```
public static View inflate(Context context, int resource,
                                                ViewGroup root) {
    LayoutInflater factory = LayoutInflater.from(context);
    return factory.inflate(resource, root);
}
```

Inflation 예제 1

- LayoutInflater 생성하기
  - View inflate하기

    inflate(int resource, ViewGroup root, boolean attachToRoot)
    - Layout XML File을 View 객체로 만들기 위해서는 LayoutInflater내의 inflater() 메소드를 사용
    - resource : view를 만들고 싶은 Layout File의 id
    - root : attachToRoot가 true일 경우 생성되는 View가 추가될 부모 View, attachToRoot가 false일 경우에는 LayoutParams 값을 설정 해주기 위한 상위 View, null로 설정할 경우 android:layout_xxxxx 값들이 무시됨
    - attachToRoot : true일 경우 생성되는 뷰를 root의 자식으로 만들고, false일 경우 root는 생성되는 View의 LayoutParam을 생성하는데 만 사용

Inflation 예제 1

- LayoutInflater 생성하기
  - View inflate하기

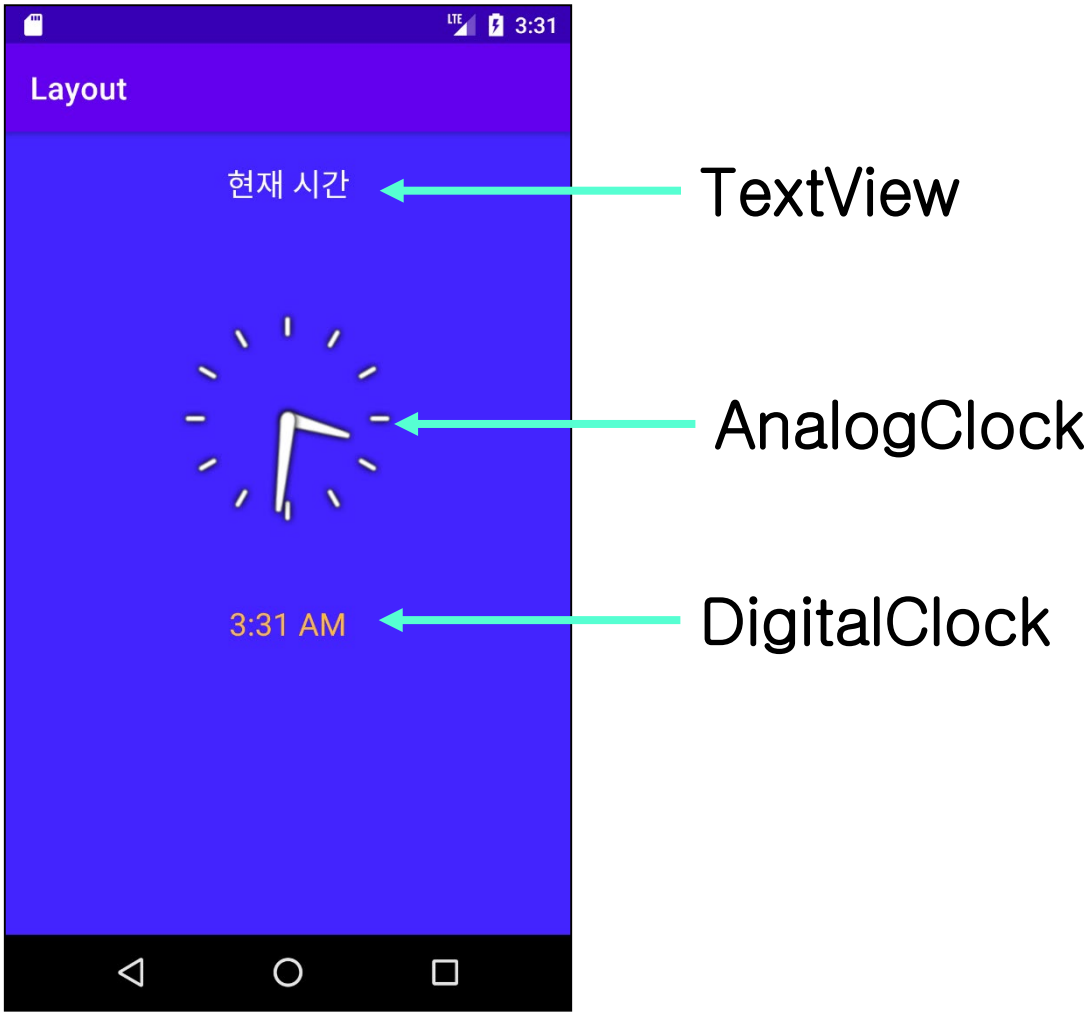
```
ViewGroup.LayoutParams params = null;
if (root != null) {
// Create layout params that match root, if supplied
        params = root.generateLayoutParams(attrs);
        if (!attachToRoot) {
            // Set the layout params for temp if we are not attaching.
            temp.setLayoutParams(params);
        }
}
...
if (root != null && attachToRoot) {
        root.addView(temp, params);
}
```

Inflation 예제 1

- XML Layout 파일에서 View를 생성할 때는 LayoutInflater를 이용해야 함
- LayoutInflater는 LayoutInflater.from(context)를 이용하여 얻을 수 있음
- LayoutInflater 객체의 inflate() 메소드를 이용하여 새로운 View를 생성 할 수 있음
- root를 지정하지 않을 경우 XML상의 최상위 Viw의 android:layout_xxxxx들은 무시됨
- attachToRoot를 true로 설정할 경우 뷰를 생성할 때 자동으로 root의 자식으로 추가됨

Layout 예제 4

- 다음과 같은 App을 만들어보자



Layout 예제 4

- 사용자 인터페이스

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center"
    android:background="#4323ff"
    tools:context=".MainActivity7">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="현재 시간"
        android:textSize="20dp"
        android:layout_marginTop="20dp"
        android:gravity="center"
        android:textColor="@android:color/white"/>
```

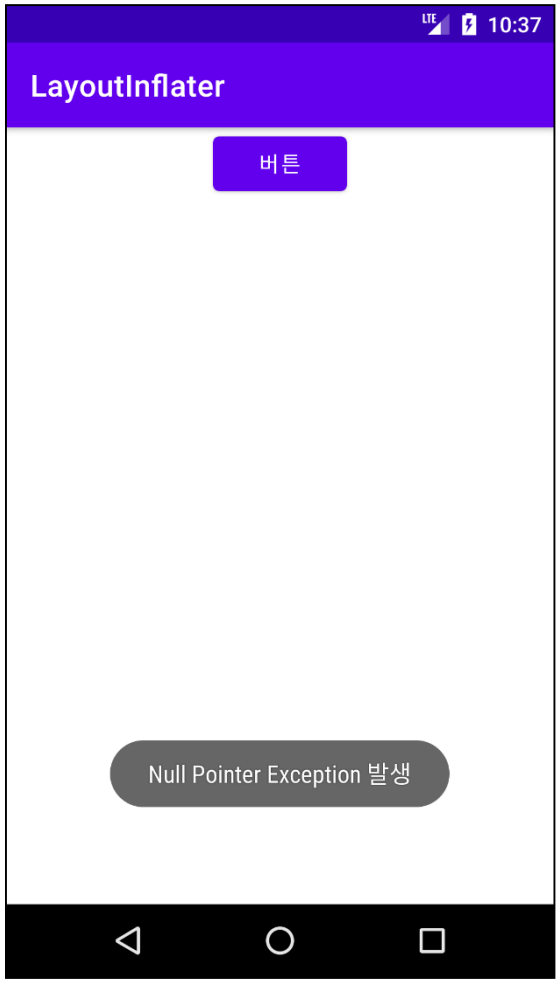
Layout 예제 4

- 사용자 인터페이스

```
    <AnalogClock
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:layout_weight="3" />

    <DigitalClock
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="20dp"
        android:layout_weight="7"
        android:textColor="@android:color/holo_orange_light"/>
</LinearLayout>
```

Layout 예제 5

- setContentView() 전에 객체를 참조한다면 ?



Layout 예제 5

- 사용자 인터페이스

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center_horizontal"
    tools:context=".MainActivity6">

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="버튼" />
</LinearLayout>
```

Layout 예제 5

- MainActivity.JAVA

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    try {
        Button button = findViewById(R.id.button);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(getBaseContext(), "버튼이 눌렸습니다.",
                                                    Toast.LENGTH_LONG).show();
            }
        });
    } catch (NullPointerException e) {
        Toast.makeText(getBaseContext(), "Null Pointer Exception 발생",
                                                        Toast.LENGTH_LONG).show();
    }
    setContentView(R.layout.activity_main);
}
```

Layout 예제 5

- 예제 Program에서 JAVA Code에서는 setContentView() 메소드를 통해 Layout File을 Memory에 Loading하여 객체로 만들기 전에 Layout에 정의 된 Button을 참조 함
- JAVA에서는 객체로 Memory에 존재하지 않는 것을 참조할 수 는 없음
- 만약에 존재하지 않는 것을 참조하려 하면 Null Pointer Exception Error가 일어나고, Program이 중지되는 심각한 오류가 발생
- setContentView() 메소드는 Activity에 보일 Layout을 설정하고 Memory에 Load하여 객체화 시켜주는 아주 중요한 메소드