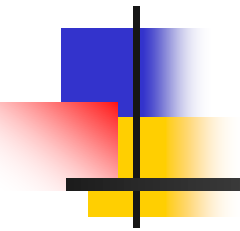


Layout

배 희호 교수
경복대학교
스마트IT과

학습목표

- Layout의 개념을 익힌다
- Activity 화면을 다양한 Layout으로 구성한다
- JAVA 코드만으로 화면을 작성해본다
- View와 Layout에 대해 이해하고, Layout을 활용, 관리하는 여러 가지 기법들에 대하여 알아본다

Layout 개요

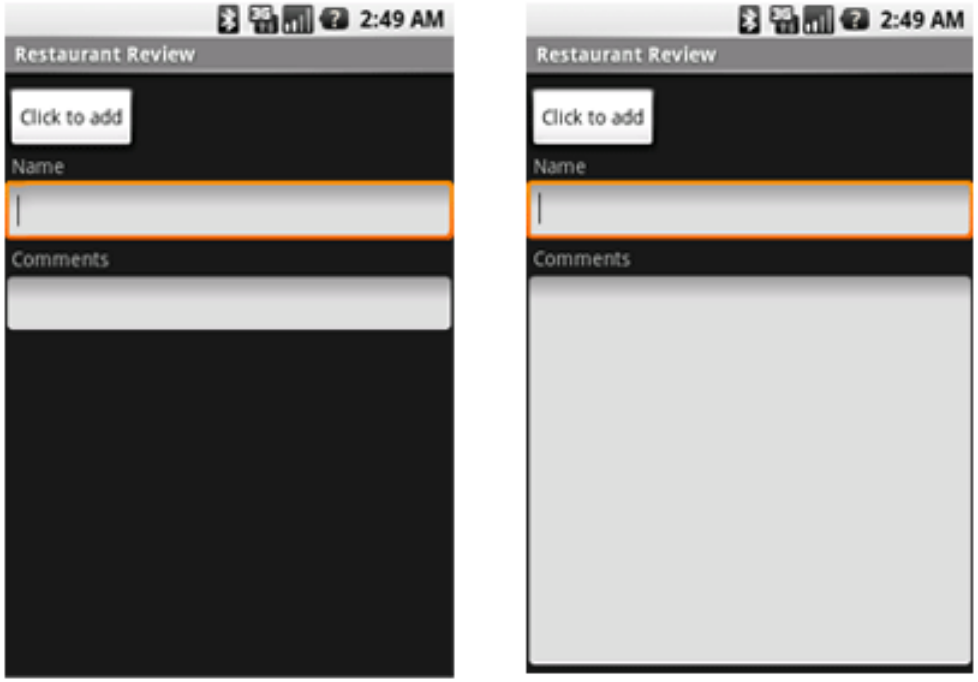
- Layout이란 무엇일까요?
  - 사전적 의미

    [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
  - Android에서의 Layout은 사용자에게 어떻게 보여줄 것인지, 어떻게 비춰질 것인지를 고려해서, UI를 만들고, 위젯들을 구성하고 배치하고, 좀 더 적합하게, 좀 더 효율적으로, 좀 더 편하게 쓸 수 있도록 해주는 것

Layout 개요

- 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 개요

- 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 보일지를 이해하고, 내 마음대로 Widget을 제어 할 수 있어야 함

Layout 개요

- 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 개요

- Layout 작성 절차
  - 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 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 개요

- 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

UI를 작성하는 3가지 방법

- XML-기반 방법 (기본적인 방법)
  - XML를 이용해서 모든 UI를 선언
  - 정적 레이아웃(Static Layout)
- JAVA-기반 방법
  - 실행 시간에 JAVA 코드로 UI를 생성
  - 동적 레이아웃(Dynamic Layout)
- Hybrid 방법
  - 최초로 보이는 UI는 XML로 UI 선언
  - 차후는 JAVA 코드로 UI 속성 수정

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="match_parent"
        android:layout_height="wrap_content"
        android:text="두번째 버튼" />

</LinearLayout>
```

선형 레이아웃 이라는 뷰 그룹을 생성

버튼 이라는 뷰를 생성

UI를 작성하는 3가지 방법

- JAVA 코드로 뷰를 생성하는 방법

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        LinearLayout layout = new LinearLayout(this);
        layout.setOrientation(LinearLayout.VERTICAL);

        Button button1 = new Button(this);
        button1.setText("첫번째 버튼");
        layout.addView(button1);

        Button button2 = new Button(this);
        button2.setText("두번째 버튼");
        layout.addView(button2);
        setContentView(layout);
    }
}
```

선형 레이아웃 생성

버튼을 선형 레이아웃에 추가

UI를 작성하는 3가지 방법

- XML과 JAVA 코드를 동시에 사용하는 방법

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <Button
        android:id="@+id/button1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="첫번째 버튼"/>
    <Button
        android:id="@+id/button2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="두번째 버튼" />
</LinearLayout>
```

버튼에
식별자를 부여

UI를 작성하는 3가지 방법

- XML과 JAVA 코드를 동시에 사용하는 방법

```
public class MainActivity extends AppCompatActivity {

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

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

        button2.setText("변경 되었어요");
        button2.setBackgroundColor(Color.BLUE);
        button1.setEnabled(false);
    }
}
```

버튼을 연결함

버튼의 속성을 변경

UI를 작성하는 3가지 방법

- XML-기반 방법 (선언적 디자인 빙식)
  - XML을 이용해서 Layout 구성
  - 선언 후에는 Runtime에 각 객체를 조정할 수 있음
  - XML Layout은 독립된 파일이기 때문에 화면이 바뀔 때마다 계속해서 변경
- JAVA-기반 방법 (절차적 디자인 방식)
  - JAVA 코딩을 통해 Layout과 버튼 등의 객체를 구성
  - JAVA 코드로 직접 작성해도 XML로 Layout을 지정하는 모든 기능을 완벽하게 처리할 수 있음
- 선언적 방식의 장점
  - 애플리케이션의 외형과 동작을 구분할 수 있음
  - 코드와 디자인의 분리
  - JAVA 코드에 비해 짧고 이해하기 쉬움
  - JAVA로 작성된 기능에 오류가 생기는 문제를 예방
  - 구성된 화면을 쉽게 가져와 사용할 수 있음

UI를 작성하는 3가지 방법

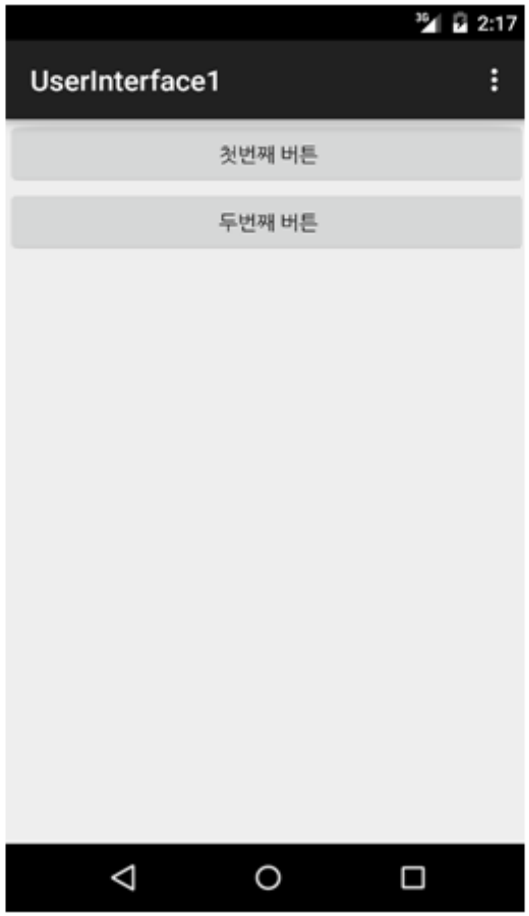
- XML-기반 방법

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>
...
```

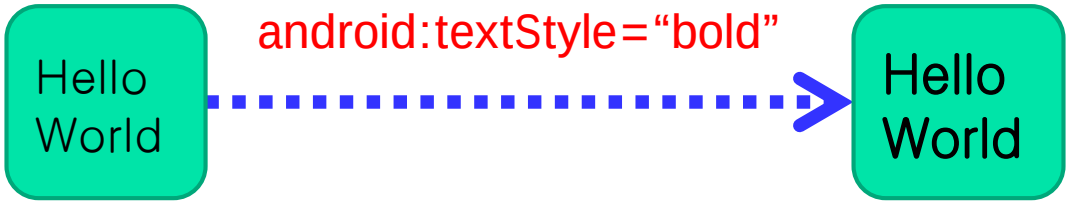
② 코드로 사용자 인터페이스 작성

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



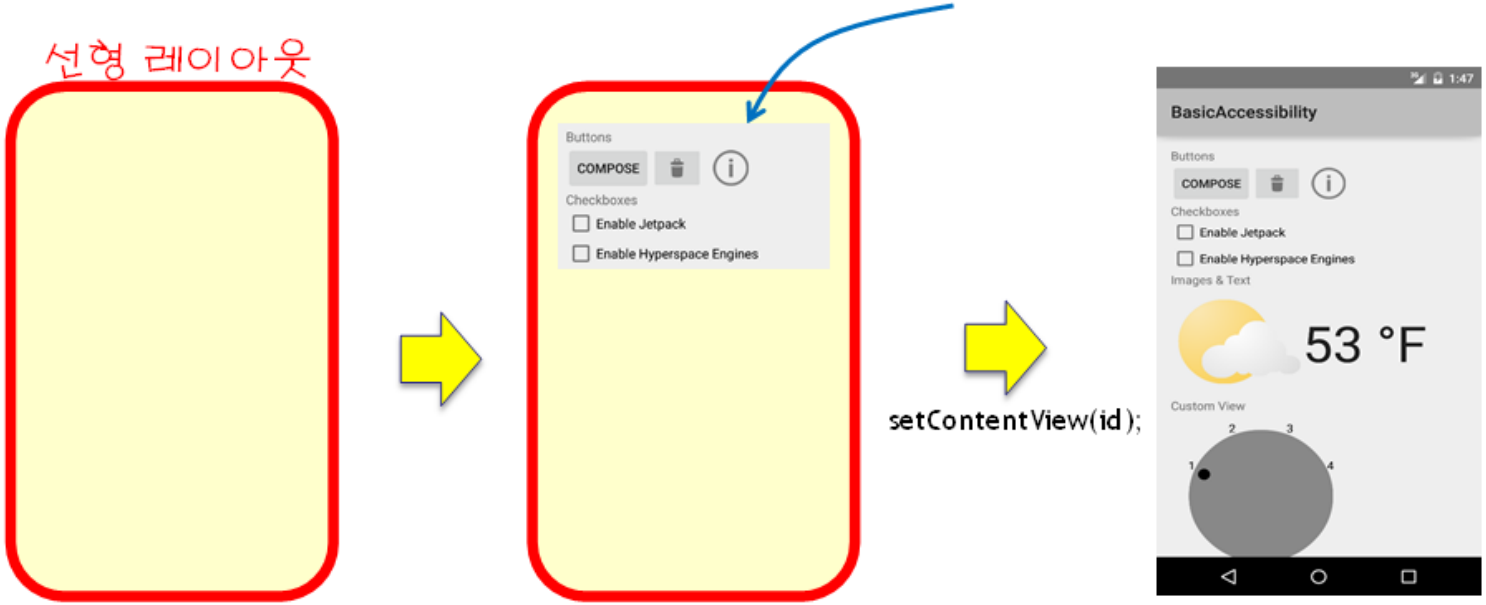
UI를 작성하는 3가지 방법

- 하이브리드(hybrid) 방법
  - 각 위젯이 서로 연결되고 포함되는 관계를 XML 형태로 정의
  - Android는 XML Layout을 리소스 파일로 인식하기 때문에 XML 레이아웃 파일은 안드로이드 내부의 [res]-[layout] 디렉토리에 보관
  - 각 XML 레이아웃 파일에 정의된 element 트리 구조를 보면 실제 화면에 나타나는 위젯과 컨테이너의 구조를 그대로 나타냄



UI를 작성하는 절차

- UI를 작성하는 절차
  1. 뷰 그룹을 생성
  2. 필요한 뷰를 추가
  3. 액티비티(Acitivity) 화면으로 설정



① 뷰그룹을 생성한다.

② 필요한 뷰들을 추가한다.

③액티비티의 화면으로 설정한다.

Layout 종류

- 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 종류

- Layout의 종류

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

그림 5-1 레이아웃의 종류

Layout 종류

레이아웃	설 명
리니어 레이아웃 (LinearLayout)	✓ 박스(Box) 모델 ✓ 사각형 영역들을 이용해 화면을 구성하는 방법 ✓ 표준 자바의 BoxLayout과 유사 ✓ 왼쪽 위부터 아래쪽 또는 오른쪽으로 차례로 배치
상대 레이아웃 (RelativeLayout)	✓ 규칙(Rule) 기반 모델 ✓ 부모 컨테이너나 다른 뷰와의 상대적 위치를 이용해 화면을 구성하는 방법 ✓ 위젯 자신이 속한 레이아웃의 상하좌우의 위치를 지정하여 배치
프레임 레이아웃 (FrameLayout)	✓ 기본 단위 모델 ✓ 하나의 뷰만 보여주는 방법 ✓ 가장 단순하지만 여러 개의 뷰를 추가하는 경우 중첩시킬 수 있으므로 뷰를 중첩한 후 각 뷰를 전환하여 보여주는 방식으로 사용할 때 유용함 ✓ 위젯들을 왼쪽 위에 일률적으로 겹쳐서 배치하여 중복해서 보이는 효과를 냄

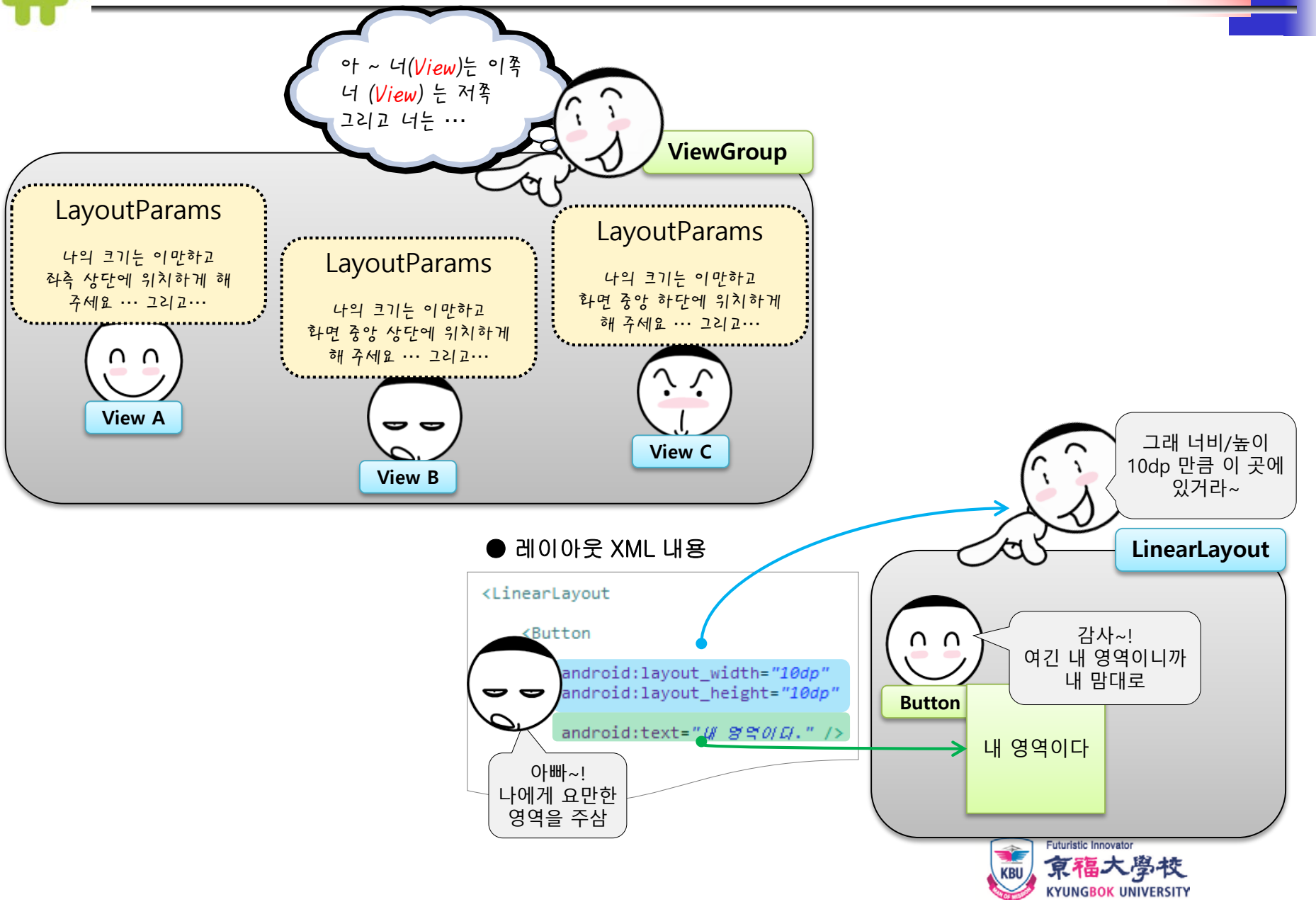
Layout 종류

레이아웃	설 명
테이블 레이아웃 (TableLayout)	✓ 격자(Grid) 모델 ✓ 격자 모양의 배열을 이용하여 화면을 구성하는 방법 ✓ HTML에서 많이 사용하는 정렬 방식과 유사하여 실용적 임 ✓ 위젯을 행과 열의 개수를 지정한 테이블 형태로 배열
그리드 레이아웃 (GridLayout)	✓ 테이블레이아웃과 비슷하지만, 행 또는 열을 확장하여 다양하게 배치할 때 더 편리
스크롤 뷰 (ScrollView)	✓ 스크롤이 가능한 컨테이너 ✓ 뷰 또는 뷰그룹이 들어갈 수 있으며 화면 영역을 넘어갈 때 스크롤 기능 제공

Layout 종류

- 레이아웃에 따라 뷰를 추가하는 방식

Layout 종류



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	부모크기만큼	가질 데이터 크기만큼	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

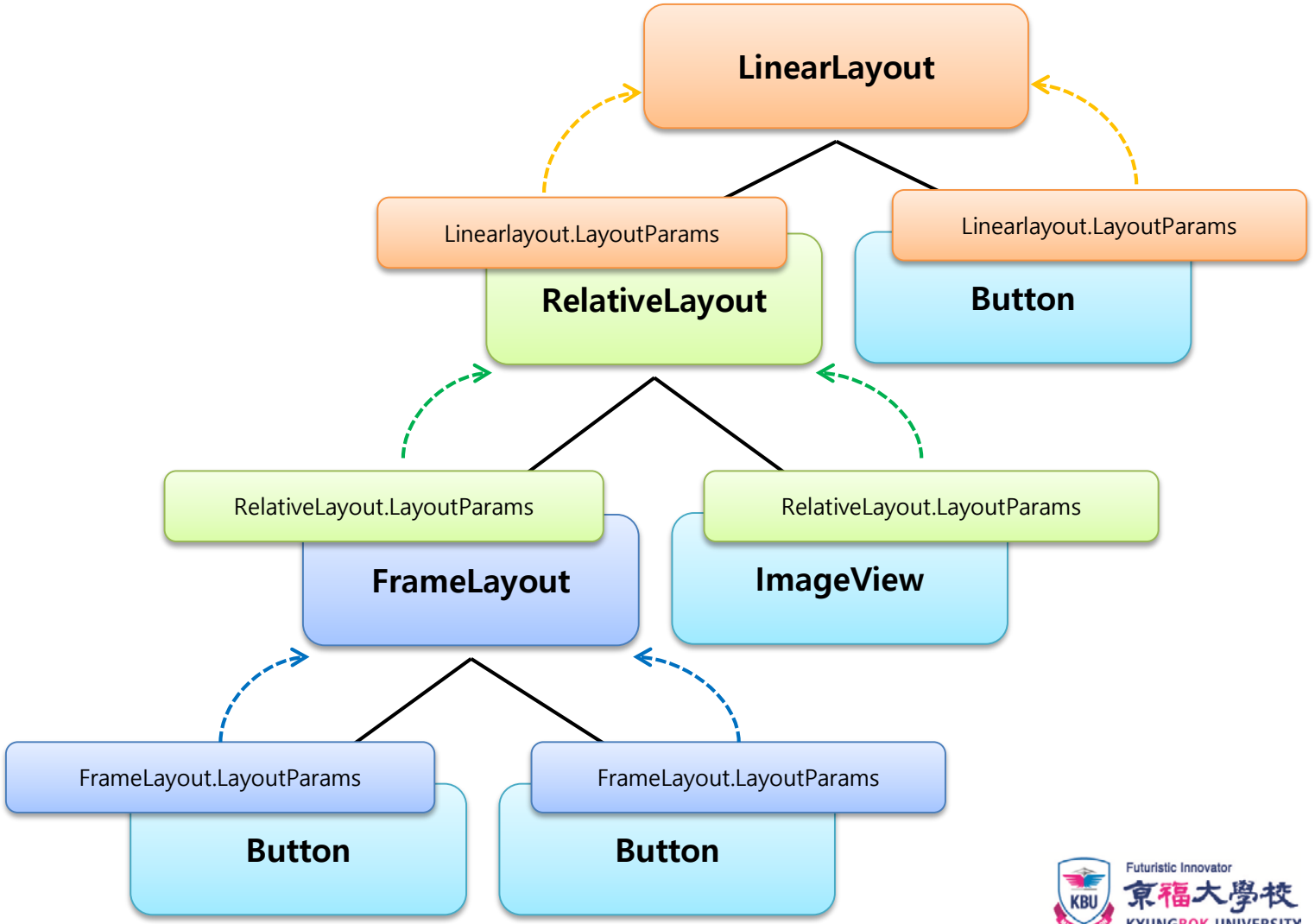
- Layout의 크기
  - 부모 크기 만큼
    (부모의 전체크기 중에서, 내가 지금 작성한 Layout보다 먼저 작성된 Layout을 제외하고 남은 크기)
  - 현재 Android SDK 버전이 올라가면서 SDK 2.2부터는 'match_parent'라는 값이 나오고 있음
    - 이 값은 기존의 'fill_parent'와 같은 값
  - 부모 크기처럼 지정할 때에는 'match_parent'라고 설정해 줌

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
  - 레이아웃 안에 배치할 위젯의 여백을 설정
- layout_weight
  - 레이아웃이 전체 화면에서 차지하는 공간의 가중 값을 설정
  - 여러 개의 레이아웃이 중복될 때 주로 사용
- baselineAligned
  - 레이아웃 안에 배치할 위젯들을 보기 좋게 정렬

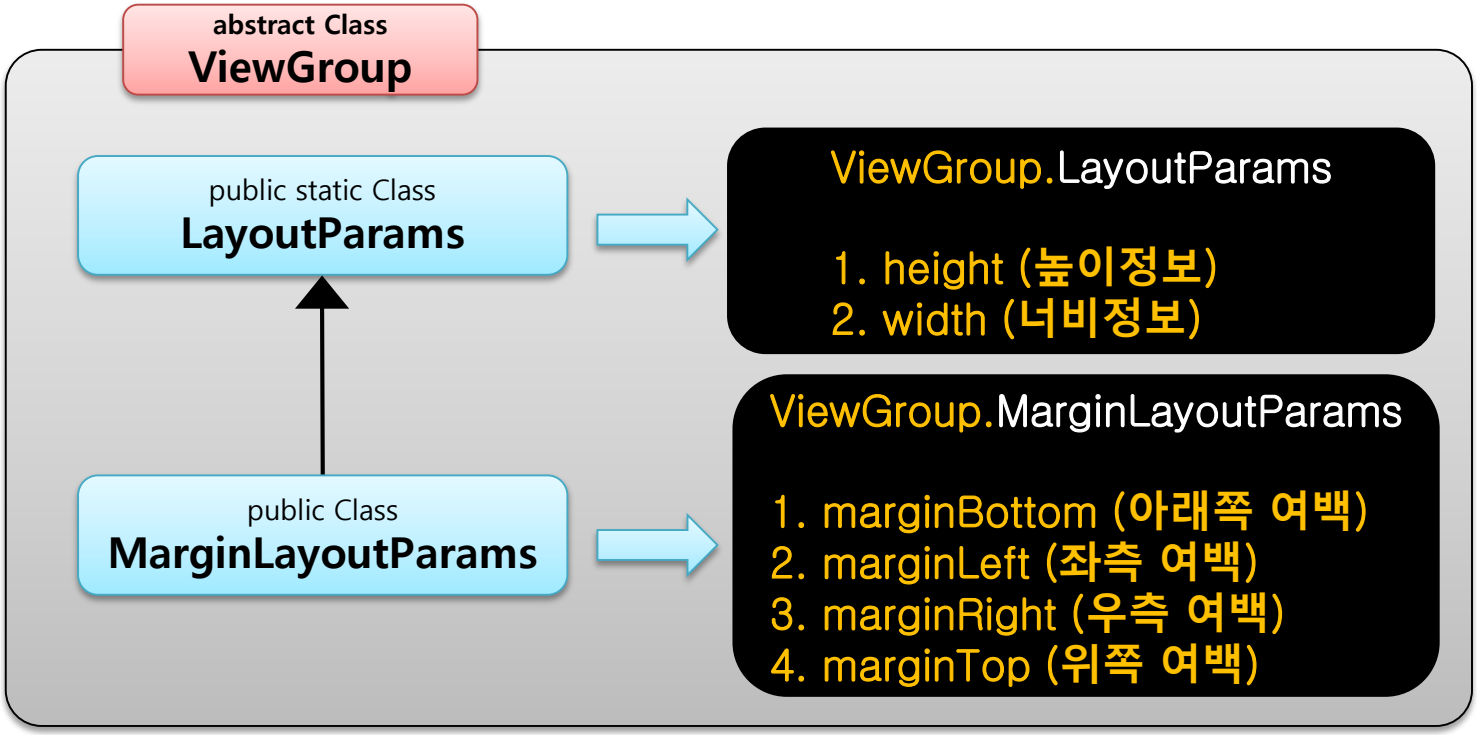
Layout 속성

- 뷰가 가지는 다양한 Layout 속성(Params) 정보



Layout 속성

- 최상위 뷰그룹의 Layout 속성(Params) 정보



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

Layout 속성

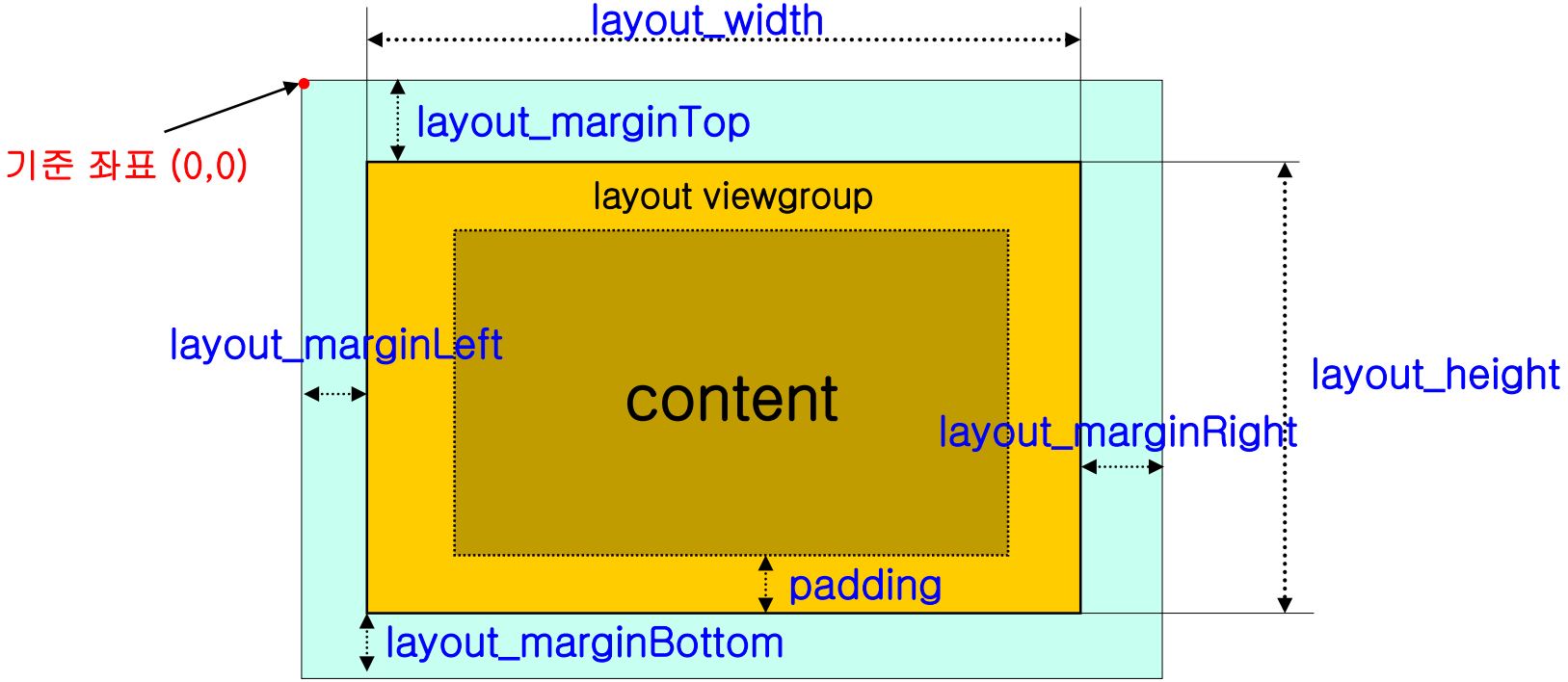
- “layout_” 속성
  - View(또는 ViewGroup)가 자신의 배치 관련 설정을 부모 Layout에게 요청하는 역할을 한다는 것
  - View에 사용되는 대부분의 속성들이 View 자체의 형태를 위한 속성이라면, "layout_"으로 시작하는 속성은 부모 Layout 내부에서 View가 가지는 크기, 여백, 위치 등을 설정하기 위한 속성인 것
  - "layout_"으로 시작하는 속성들을 사용한다고 해서, 그 속성 값이 View의 배치에 그대로 적용되는 것은 아님
  - Layout 내에 있는 다른 View 위젯들의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

Layout 속성

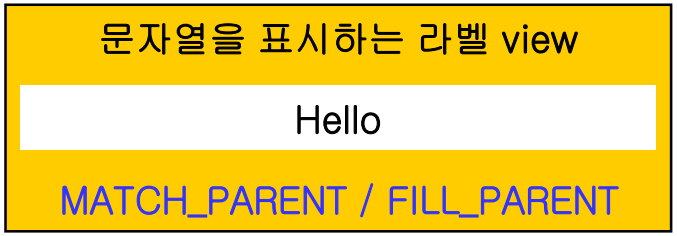
- layout_margin, padding
  - Layout에 자식 View 위젯들이 표시될 때, Layout과 View 위젯 사이 또는 인접한 두개의 View 위젯 사이에 여백을 위한 공간이 존재하지 않으면, 앱이 보여주고자 하는 내용의 가독성과 사용 편의성이 현저히 떨어지게 됨

Layout 속성

- Layout의 좌표와 용어

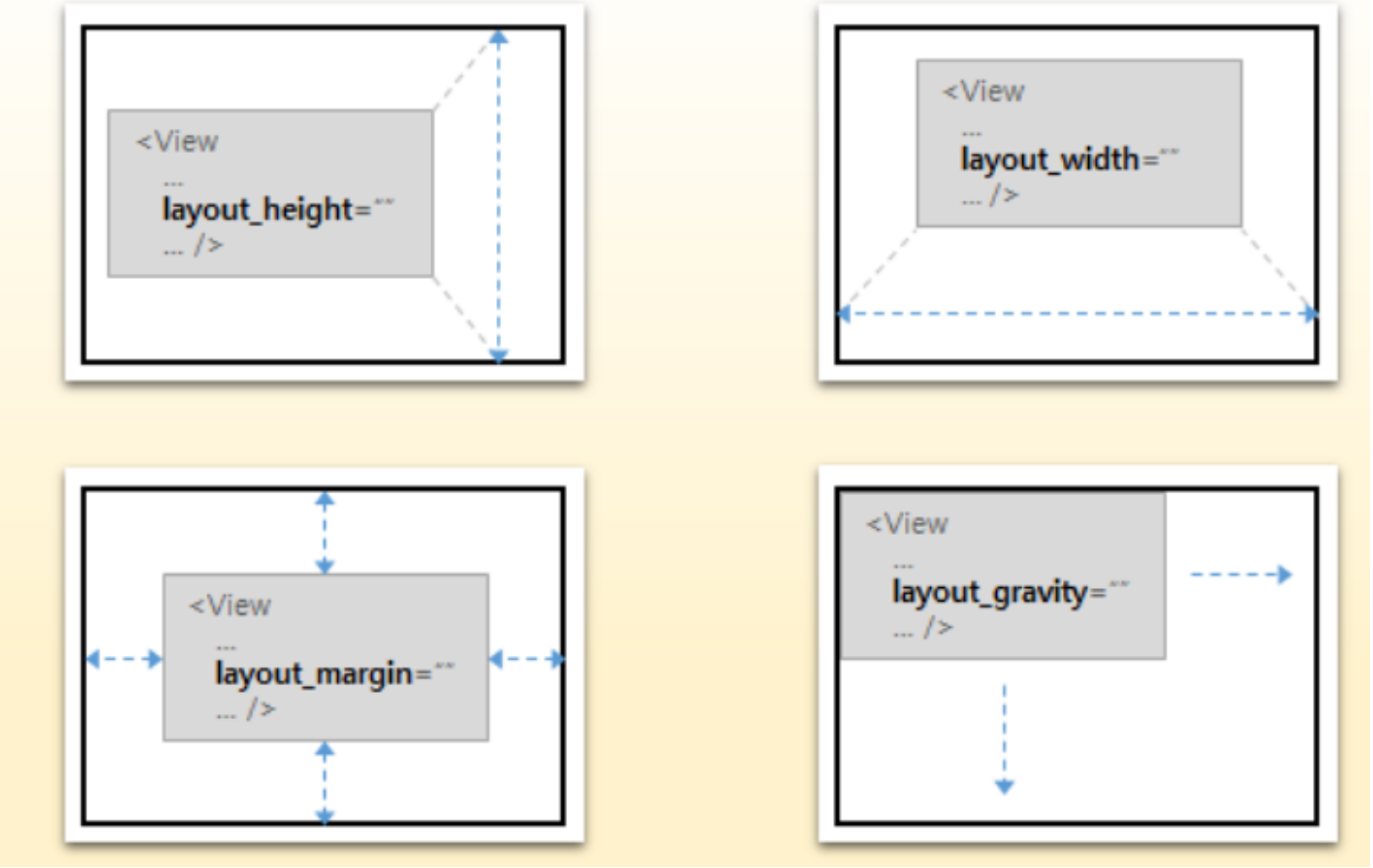


컨텐츠를 표시하는 데 충분한 크기로



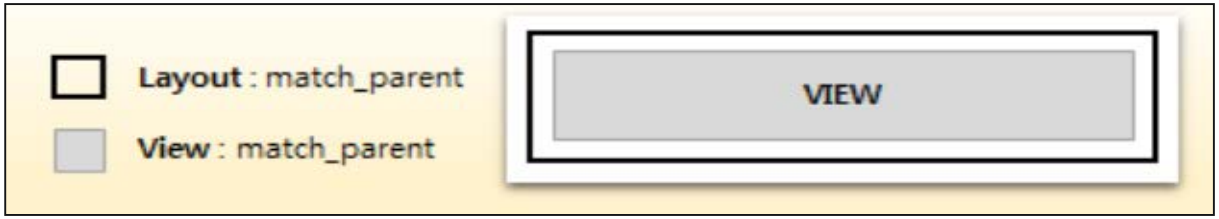
부모 객체와의 패딩(여백)을 제외한 나머지 공간을 차지

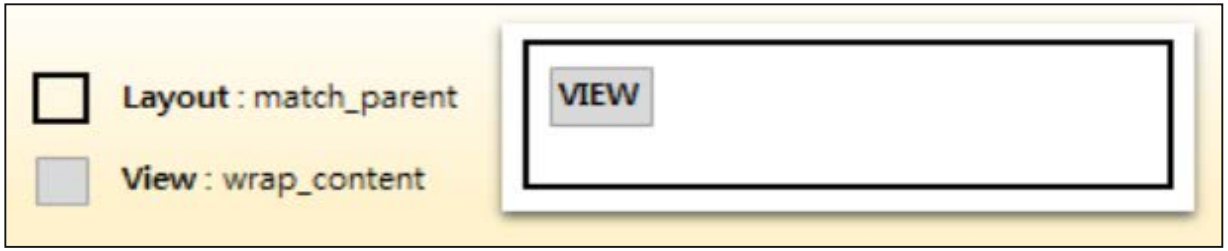
Layout 속성


<View
...
layout_height=""
... />
<View
...
layout_width=""
... />
<View
...
layout_margin=""
... />
<View
...
layout_gravity=""
... />

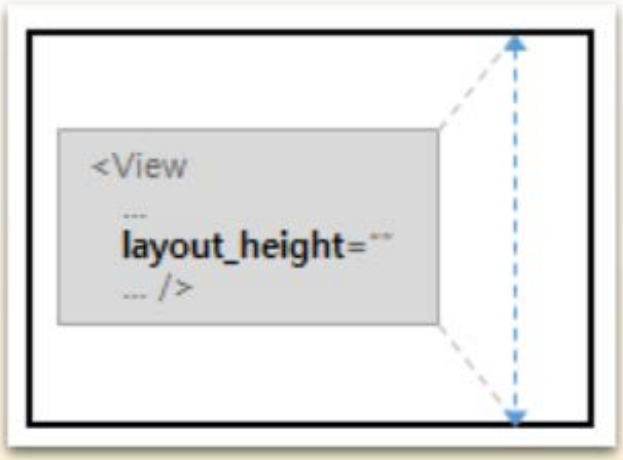
Layout 속성

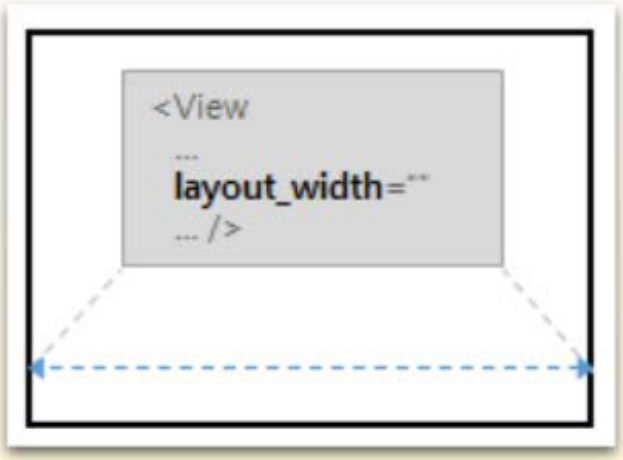
- Layout(검은 실선 영역)과 자식 View 위젯과의 관계

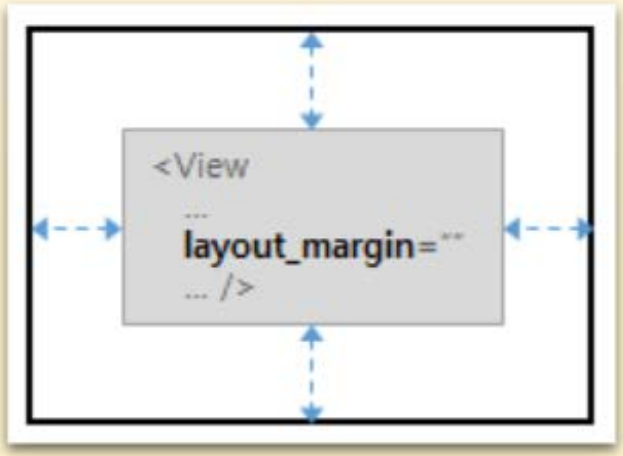


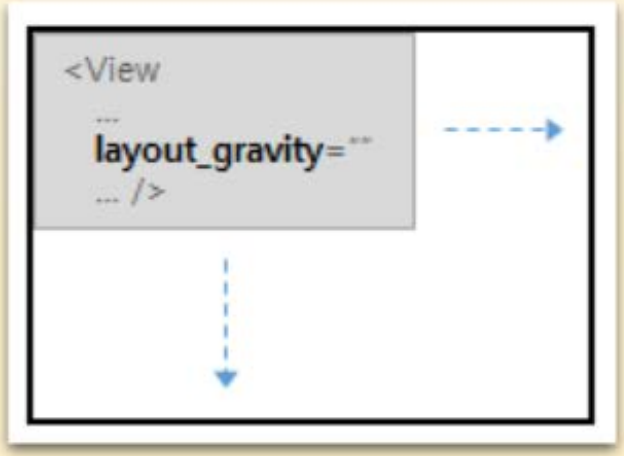


Layout 속성



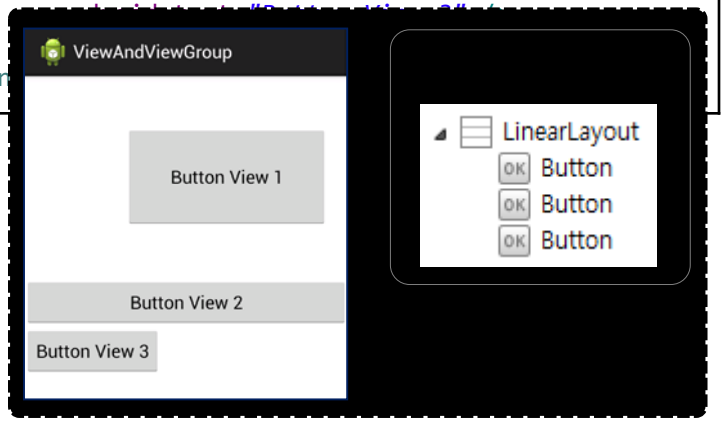


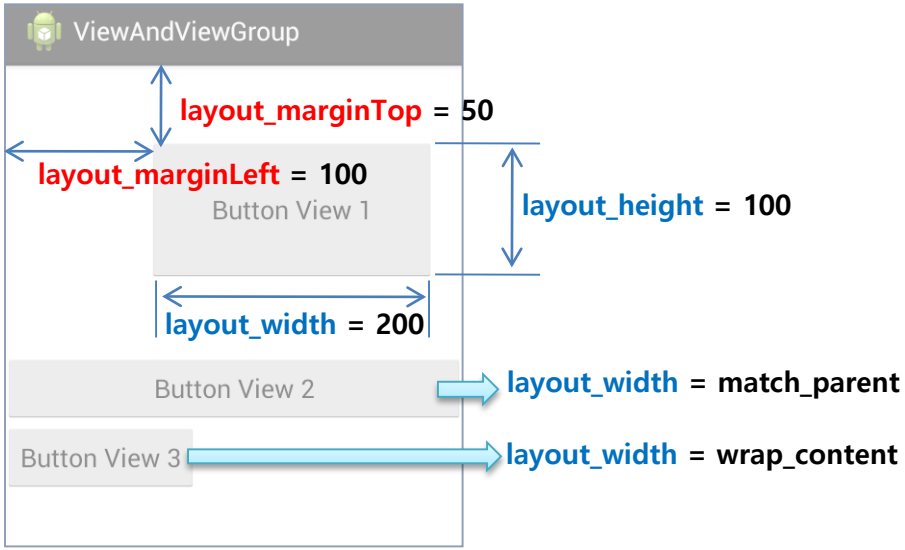


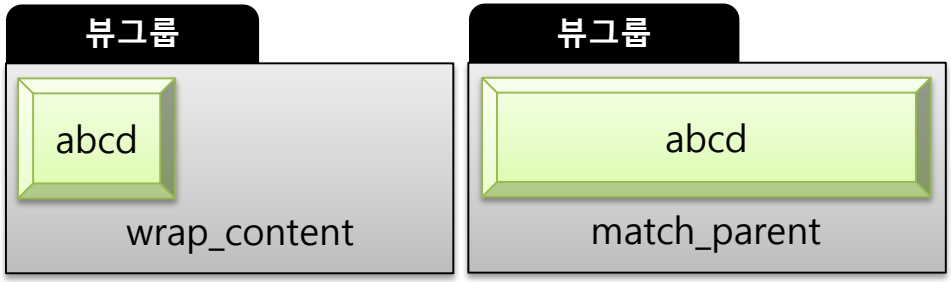


Layout 속성

```
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
```

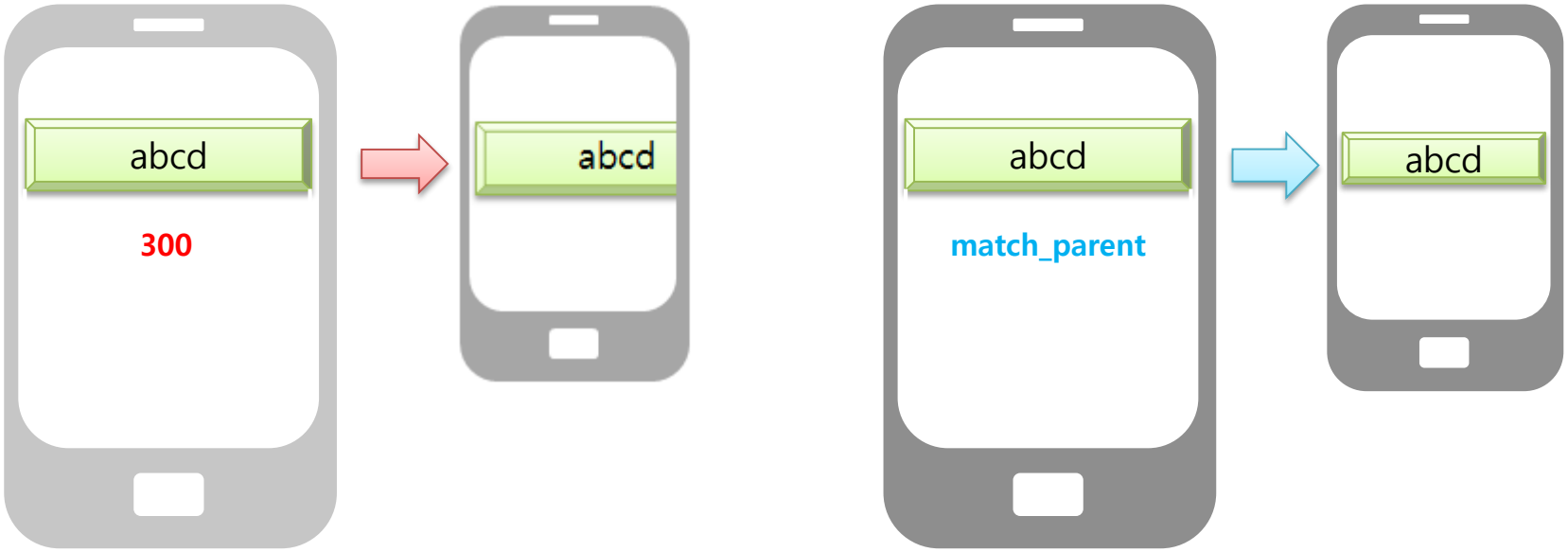






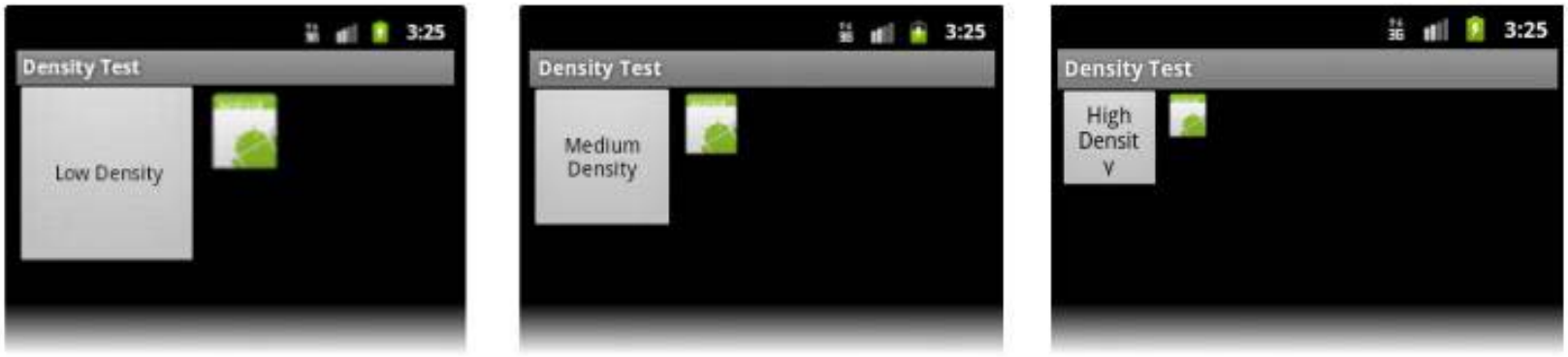
Layout 속성

- match_parent는 부모 뷰그룹에 크기를 맞추기 때문에 화면 크기가 다른 단말에서도 유연하게 레이아웃을 유지할 수 있다

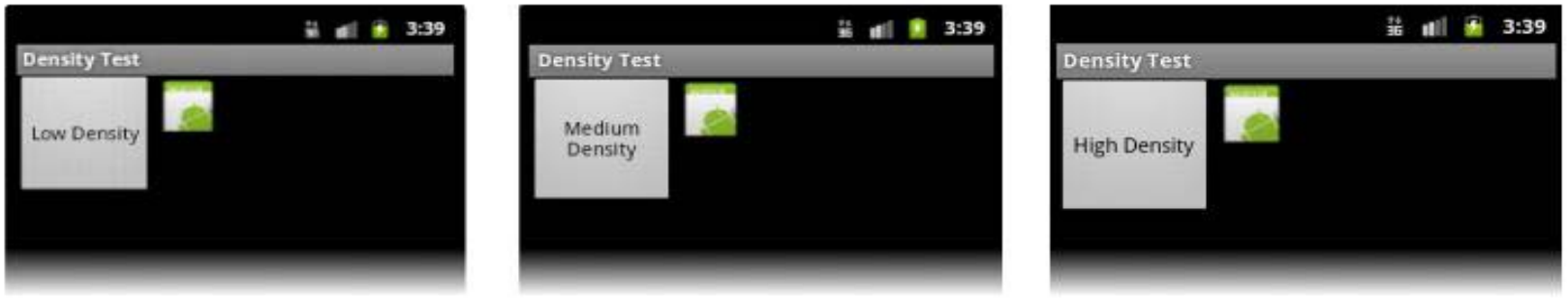


Layout 속성

- DIP(Density independent pixel) 사용
  - PX 사용

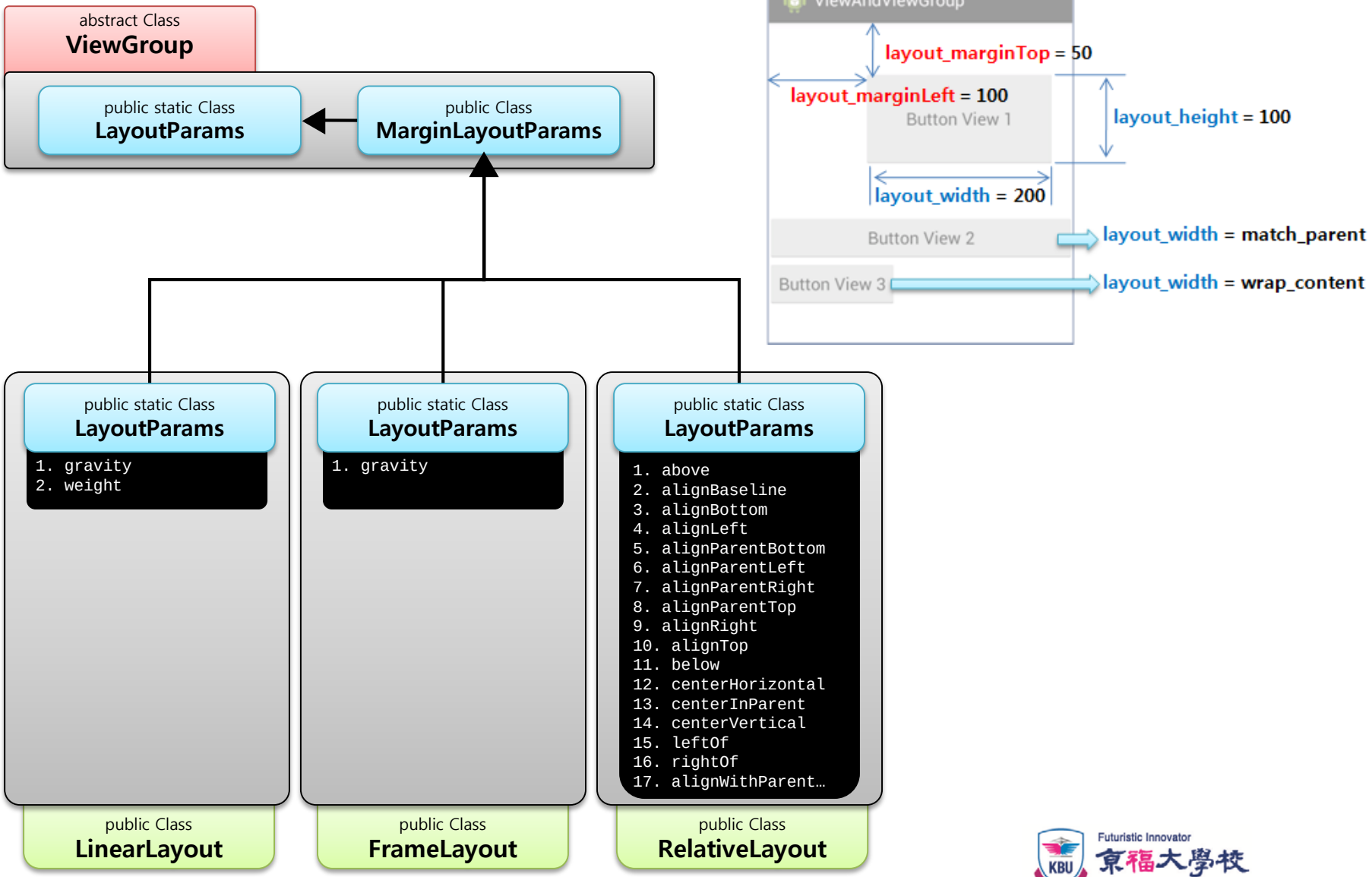


  - DIP 사용



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

Layout 속성



XML 기반 Layout

- 각 Widget이 서로 연결되고 포함되는 관계를 XML 형태로 정의
- Android는 XML Layout을 Resource 파일로 인식함
- XML Layout 파일은 프로젝트 내부의 res/layout Directory 아래 보관
- 각 XML Layout 파일에 정의된 엘리먼트 트리 구조는 실제 화면에 나타나는 Widget과 Container의 구조를 그대로 나타냄
- XML 엘리먼트에 지정된 속성은 Widget 속성에 연결돼 Widget이 표현되는 방법이나 Container가 동작하는 방법 등을 정의
- 예) Button 엘리먼트에 android:textStyle="bold" 라고 지정되 있으면 해당 버튼이 화면에 표시될 때 굵은 글꼴을 사용한다는 의미

XML 기반 Layout

- XML로 레이아웃을 지정하는 이유
  - JAVA 코드와의 독립성
    - JAVA 코드와 XML Layout 파일은 서로가 변경되더라도 서로에게 영향을 끼치지 않음
  - GUI 화면 디자인 프로그램에서 화면을 좀 더 쉽게 생성하고 관리하겠다는 목표

XML 기반 Layout

- XML Layout 파일 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 파일의 루트 엘리먼트는 Android XML 네임스페이스를 지정해야 함

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

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

XML 기반 Layout

- XML Layout 파일 구조
  - android:layout_width / android:layout_height 속성
    - 자신을 포함하는 부모(parent) 엘리먼트의 폭과 너비를 그대로 사용
  - 엘리먼트 ID
    - JAVA코드에서 호출해 사용하려는 모든 엘리먼트들은 반드시 android:id 속성으로 id를 지정해야 함
    - 일반적으로 @+id/... 와 같은 아이디를 지정
    - ...부분에는 중복되지 않는 유일한 문자열을 지정해야 함
    - 예) android:id="@+id/button”
    - 안드로이드에서는 @android:id/...와 같이 특별한 형태의 android:id값을 사용하기도 함

XML 기반 Layout

- JAVA 코드와 연결하는 방법
  - 필요한 위젯과 컨테이너를 구성해 main.xml 파일로 저장하고 res/layout 디렉토리에 보관
  - 액티비티의 onCreate() 메소드에서 다음과 같은 코드 한 줄만 적어주면 작성한 레이아웃을 그대로 불러옴

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

XML 기반 Layout

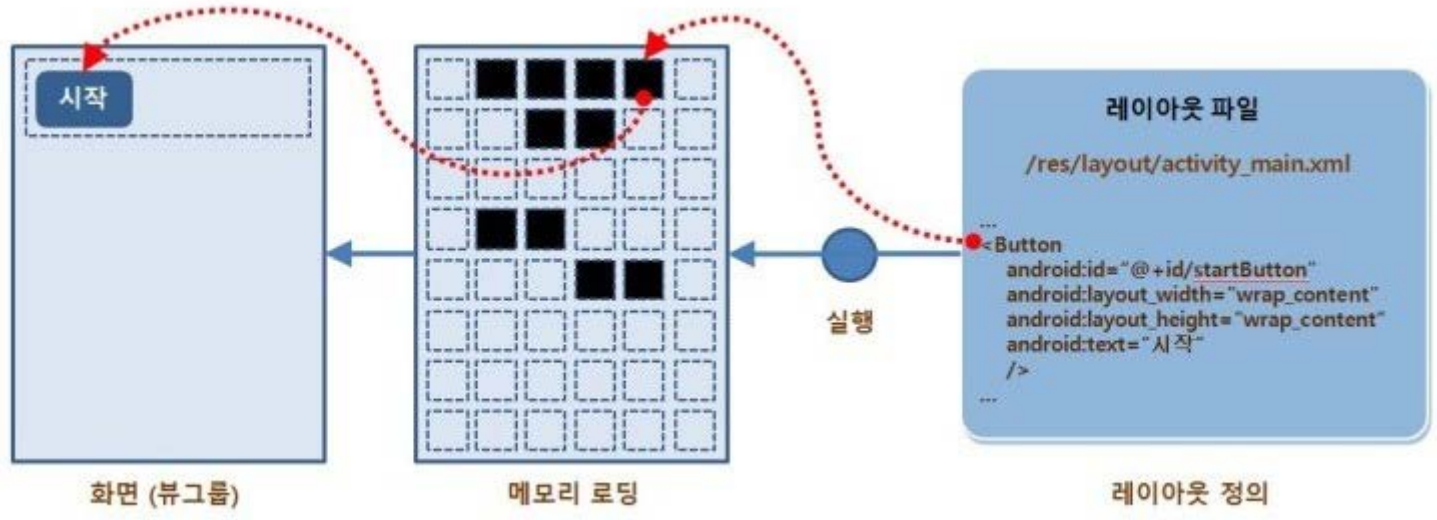
- JAVA 코드와 연결하는 방법
  - 엘리먼트 가운데 id를 지정한 항목 찾아오기
    - findViewById() 메소드를 사용
    - 메소드 인자로는 찾고자 하는 Widget의 숫자 ID를 넘겨야 함
    - Widget의 숫자 ID는 Android가 자동으로 생성해 R 클래스에 반영, R.id.something으로 사용(something 부분에 Widget 이름)
  - R.java
    - XML Layout 파일을 Android 빌드 시스템이 분석해 JAVA 코드에서 호출해 쓸 수 있도록 자동 생성한 자바 코드 파일
    - 모든 Layout 정보는 R.layout 변수를 통해 접근, 레이아웃 파일 이름으로(main.xml파일의 레이아웃은 R.layout.main) 접근

XML 기반 Layout

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

LayoutInflater 사용하기

- LayoutInflater란?



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

LayoutInflater 사용하기

- LayoutInflater란?
  - 보통 JAVA 코드에서 View, ViewGroup을 사용하거나, Adapter의 getview() 또는 Dialog, Popup 구현 시 배경화면이 될 Layout을 만들어 놓고 View의 형태로 반환 받아 Acitivity에서 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 Activity 위에 보여줌
  - 사용자의 화면에 보여지는 것들은 Activity 위에 있는 View

LayoutInflater 사용하기

- Android에서 Layout XML 파일을 View 객체로 만들기 위해서는 LayoutInflater를 이용
  - Context.getSystemService()

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

LayoutInflater 사용하기

- Android에서 Layout XML 파일을 View 객체로 만들기 위해서는 LayoutInflater를 이용
  - LayoutInflater.from() 메소드

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

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

LayoutInflater 사용하기

- Activity.getLayoutInflater()

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

  - Activity에서는 LayoutInflater를 쉽게 얻어올 수 있도록 getLayoutInflater() 메소드를 제공
  - Activity의 윈도우에 있는 getLayoutInflater()로 포워딩

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

LayoutInflater 사용하기

- View 팩토리 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);
}
```