이벤트 처리 (1)

Mobile Software
2022 Fall

What to do next?

- 이벤트 핸들러를 구현하는 3가지 방법
  - 중첩 클래스
  - Anonymous class
  - Activity 클래스에서 직접 상속
  - 실습 1: 다양한 단위 변환 방식 적용
  - 실습 2: 한 개의 view에 대한 2개의 이벤트 처리

- 이벤트 종류
  - 실습 3: single touch 이벤트
  - 실습 4: multi-touch 이벤트

- 강의 노트에 포함된 코드: 5-1.소스코드.hwp

Create a New Project

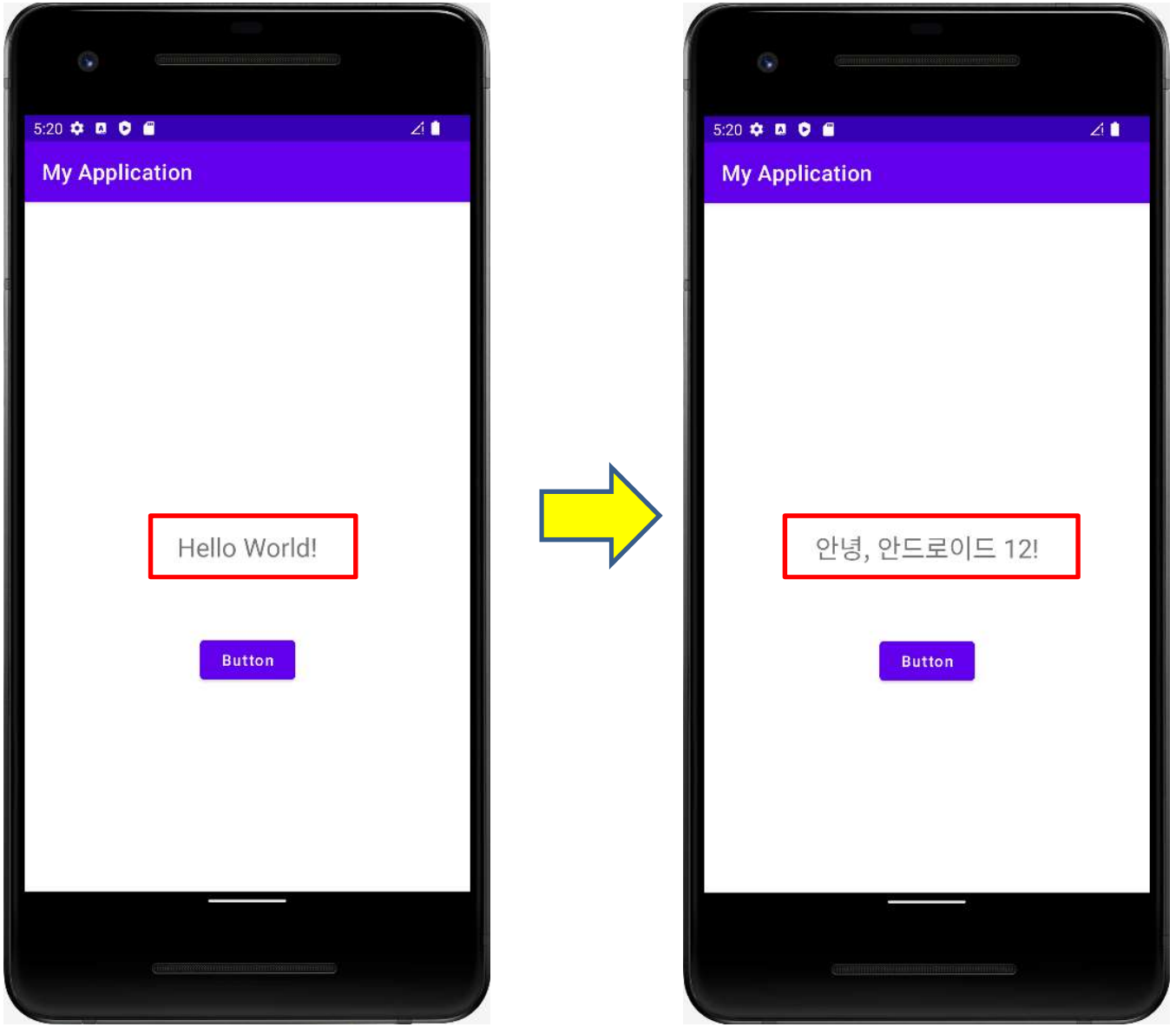
- 새 프로젝트 생성
  - **Project name** : **My Application**
  - **Package name : com.example.myapplication**
  - **Activity** : **Empty Activity**
  - **Activity name** : **MainActivity.kt**
  - **Layout name** : **activity_main.xml**

- **자동 생성된 XML 파일**의 root view는 **ConstraintLayout**

How to build an event handler

- **delegation** (위임) **model**
  - 발생한 이벤트를 view 객체에 전달
    - 이벤트 처리를 view 객체에게 맡김 ➔ **위임**
  - View 객체는
    - 어떤 이벤트가 발생하는지 알고 있음 ➔ **등록**
    - 이벤트를 어느 메소드가 처리할지 알고 있음. ➔ **콜백 메소드**
- 구현
  - **Step 1 (등록) – View 객체가 이벤트 listener를 등록**
    - **setOnXXXListener**
  - **Step 2 (메소드 구현) – 이벤트의 listener 구현**
    - Listener 인터페이스를 상속받은 클래스 정의
      - 이벤트를 처리하는 추상 메소드(callback 메소드) 구현

When the button is clicked, the string is changed

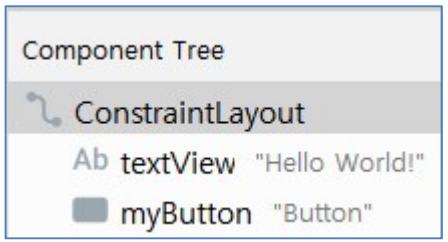


3 ways to build an event handler

- **방법 1**
  - Listener 인터페이스를 상속받는 **내부 클래스** 정의
- **방법 2**
  - **Anonymous class** (**무명 클래스**)
    - 클래스 이름이 없음 ➔ 람다 식
- **방법 3**
  - **Activity 클래스에서** Listener 인터페이스를 **상속**
    - 이벤트 핸들러가 Activity 클래스의 멤버 메소드

Layout : TextView + Button

```
<Button
    android:id="@+id/myButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="64dp"
    android:text="Button"
    android:textAllCaps="false"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textView
```



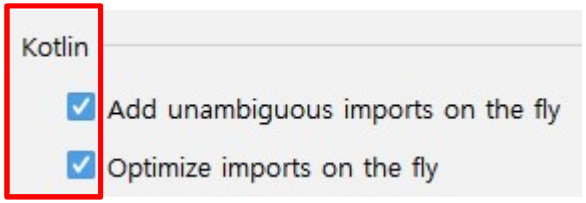
소스코드 – 1쪽

Look at the import statements!

```
super.onCreate(savedInstanceState)
setContentVi   android.widget.Button? Alt+Enter

val btnView = findViewById<Button>(R.id.btnView)
btnView.setOnClickListener (this)
```

Alt + Enter ➔
필요한 클래스를
자동으로 import

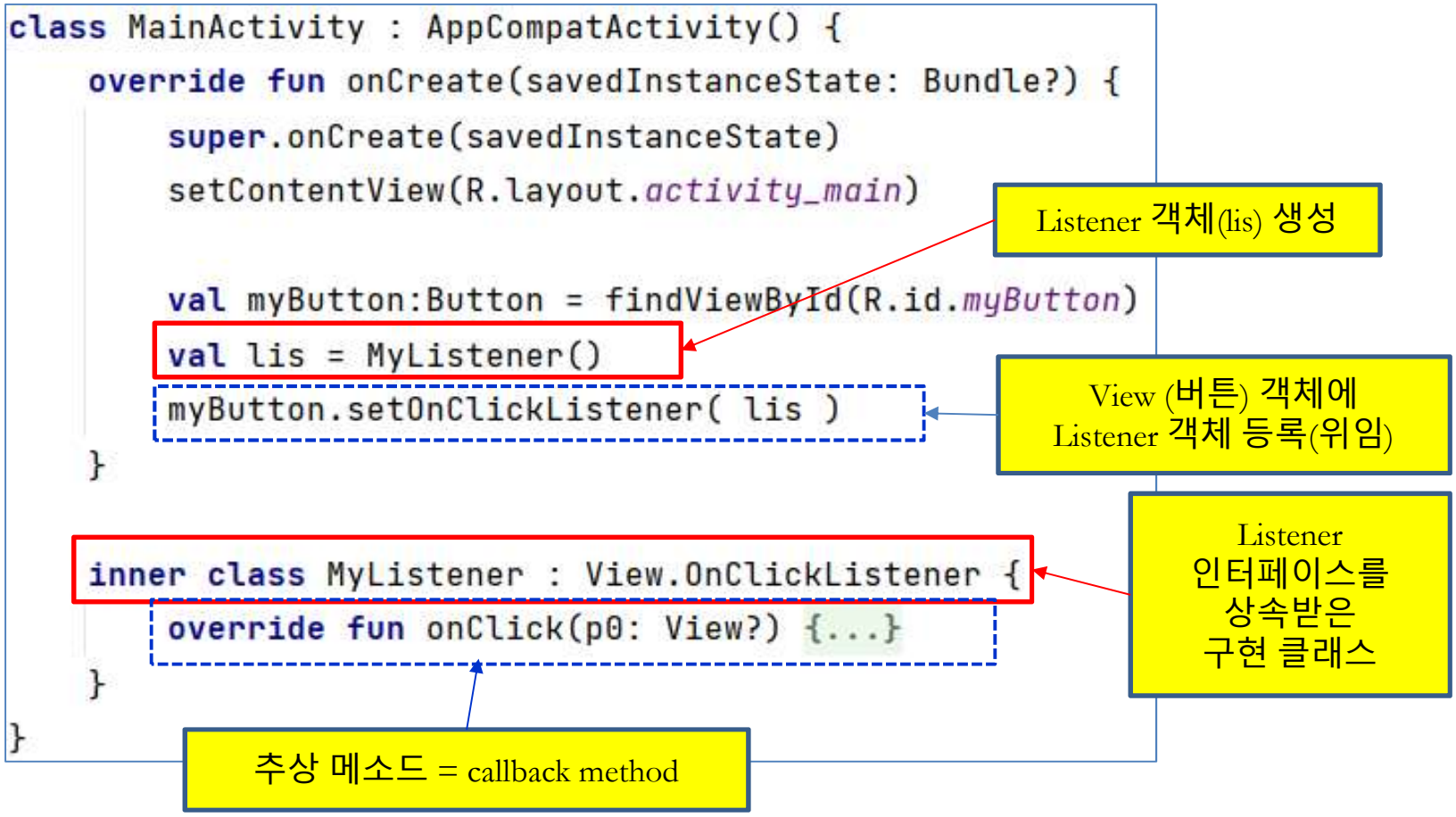
Settings > Editor >
General > Auto import




```
package com.example.myapplication

import android.os.Bundle
import android.view.View
import android.widget.Button
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity
```

Import 문의 마지막 단어가
클래스 이름 ➔ type으로 사용

방법 1: 내부 클래스로 이벤트 핸들러 정의(1/2)

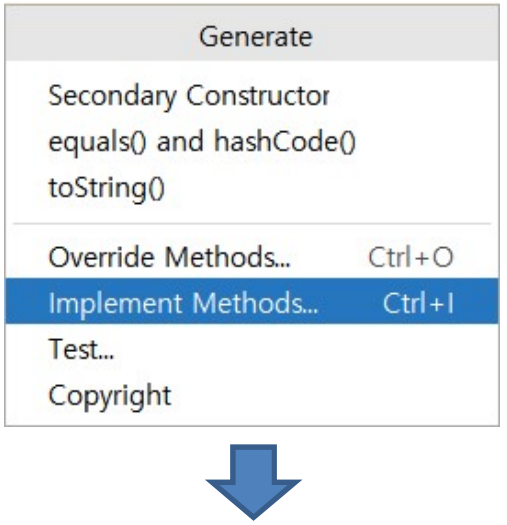


인터페이스에서 선언한 추상 메소드 자동 생성

1. BtnListener 클래스 = interface를 상속받은 클래스

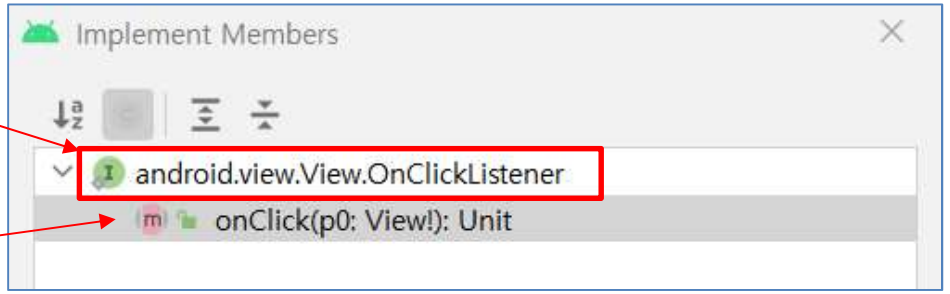
```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }

    inner class MyListener : View.OnClickListener {

    }
}
```

2. 오른쪽 버튼 > **generate**…



OnClickListener 는 Interface

3. 구현 상속을 받으면
추상 메소드 (abstract method)인
onClick 메소드를
반드시 구현해야 함 !!!



방법 1: 내부 클래스로 이벤트 핸들러 정의(2/2)

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

        val myButton: Button = findViewById(R.id.myButton)
        val lis = MyListener()
        myButton.setOnClickListener( lis )
    }

    inner class MyListener : View.OnClickListener {
        override fun onClick(p0: View?) {
            val s:String = "안녕, 안드로이드 12!"
            val textView: TextView = findViewById(R.id.textView)
            textView.text = s
        }
    }
}
```

Listener 객체 등록

```
myButton.setOnClickListener( MyListener() )
```

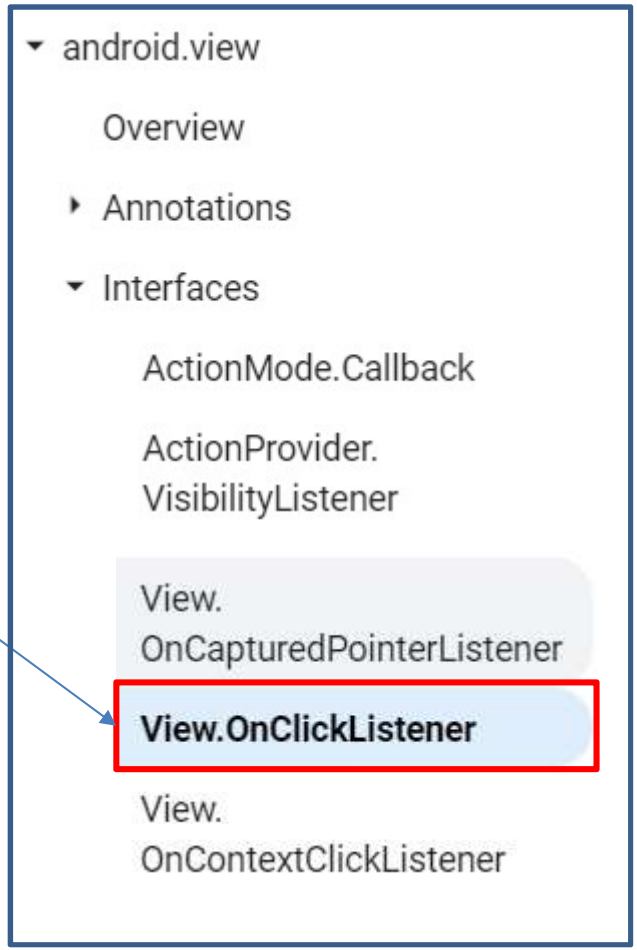
이벤트 핸들러
클래스 이름은 대문자로 시작

소스코드 – 2쪽

잠깐! View. OnClickListener

```
package com.example.myapplication

import android.os.Bundle
import android.view.View
import android.widget.Button
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity
```

OnClickListener 는
View 클래스의 멤버 인터페이스



클래스 상속과 구현 상속

클래스 상속 class A : **B( )**
➔ B는 **class**
➔ B 뒤에 반드시 **괄호가 있음** (생성자를 호출)

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

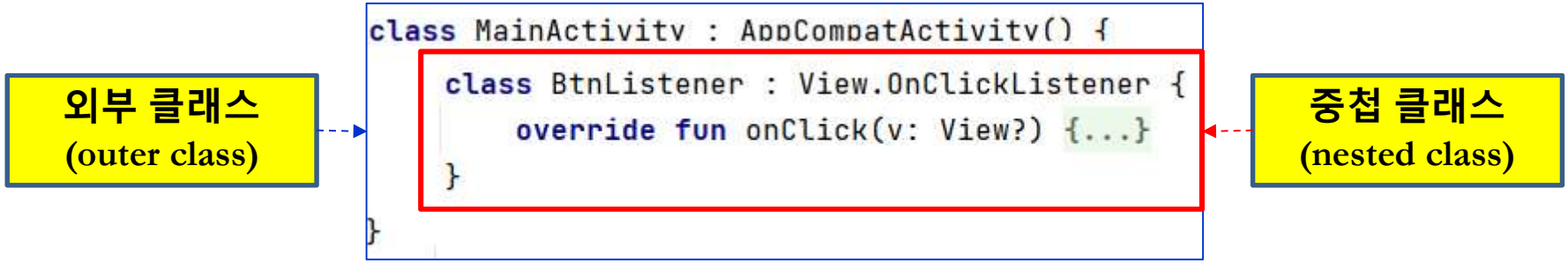
구현 상속 class **A** : **B**
➔B는 **interface** (B 뒤에 **괄호가 없음**)
➔ interface 는 추상 메소드(abstract method)를 갖고 있음.
➔ 구현 상속받은 클래스 **A**는 **추상 메소드를 반드시 구현해야 함**.

```
class BtnListener : View.OnClickListener {
    override fun onClick(v: View?) {...}
}
```

abstract method

중첩 클래스와 내부 클래스 (1/2)

- 같은 점
  - 클래스 안에 정의한 클래스
- 다른 점
  - **중첩 클래스 (nested class)**
    - 키워드 없이 사용.
    - 외부 클래스(outer class) 속성을 참조할 수 없음.
  - **내부 클래스 (inner class)**
    - 클래스 이름 앞에 키워드 **inner**를 붙임
    - **외부 클래스 속성을 참조할 수 있음**.



중첩 클래스와 내부 클래스 (2/2)

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

    class MyListener : View.OnClickListener {
        override fun onClick(p0: View?) {
            val s:String = "안녕, 안드로이드 12!"
            val textView: TextView = findViewById(R.id.textView)
            textView.text = s
        }
    }
}
```

중첩 클래스
(nested class)

Unresolved reference: findViewById

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

    inner class MyListener : View.OnClickListener {
        override fun onClick(p0: View?) {
            val s:String = "안녕, 안드로이드 12!"
            val textView: TextView = findViewById(R.id.textView)
            textView.text = s
        }
    }
}
```

내부 클래스
(inner class)

소스코드 – 2쪽

Quiz 1: 아래 코드는 에러가 있을까? 없을까?

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

        class MyListener : View.OnClickListener {
            override fun onClick(p0: View?) {
                val s:String = "안녕, 안드로이드 12!"
                val textView: TextView = findViewById(R.id.textView)
                textView.text = s
            }
        }

        val myButton: Button = findViewById(R.id.myButton)
        myButton.setOnClickListener( MyListener() )
    }
}
```

소스코드 – 3쪽

Quiz 2: 아래 코드는 왜 에러가 발생할까?

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

        val textView: TextView = findViewById(R.id.textView)
        val myButton:Button = findViewById(R.id.myButton)
        myButton.setOnClickListener( MyListener() )
    }

    inner class MyListener : View.OnClickListener {
        override fun onClick(p0: View?) {
            val s:String = "안녕, 안드로이드 12!"
            textView.text = s
        }
    }
}
```

소스코드 – 4쪽

방법 2: Anonymous class

- 클래스 body는 정의하지만, **이름이 없는(무명)** 클래스
  - 클래스 정의와 동시에 객체를 생성
  - 무명 클래스를 정의할 때는 **object** 키워드 사용

```
val myButton:Button = findViewById(R.id.myButton)
myButton.setOnClickListener( inner class MyListener : View.OnClickListener {
    override fun onClick(p0: View?) {
        val s:String = "안녕, 안드로이드 12!"
        val textView: TextView = findViewById(R.id.textView)
        textView.text = s
    }
} )
```

```
myButton.setOnClickListener( object : View.OnClickListener {
    override fun onClick(p0: View?) {
        val s:String = "안녕, 안드로이드 12!"
        val textView: TextView = findViewById(R.id.textView)
        textView.text = s
    }
} )
```

소스코드 – 5쪽

방법 2: 람다 식 변환

```
myButton.setOnClickListener( object : View.OnClickListener {
    override fun onClick(p0: View?) {
        val s:String = "안녕, 안드로이드 12!"
        val textView: TextView = findViewById(R.id.textView)
        textView.text = s
    }
} )
```

```
myButton.setOnClickListener {  it: View!
    val s: String = "안녕, 안드로이드 12!"
    val textView: TextView = findViewById(R.id.textView)
    textView.text = s
}
```

Concise version
람다 식으로 표현

소스코드 – 6쪽

람다 식 변환 과정

```
myButton.setOnClickListener( object : View.OnClickListener {
    override fun onClick(v: View?) {
        val s:String = "안녕, 안드로이드 12!"
        textView.text = s
    }
})
```

1. Lambda 식 변환

```
myButton.setOnClickListener({ v:View? ->
    val s:String = "안녕, 안드로이드 12!"
    textView.text = s
})
```

2. 함수 인자가 함수 ➔ 괄호 밖으로 이동.

```
myButton.setOnClickListener() { v:View? ->
    val s:String = "안녕, 안드로이드 12!"
    textView.text = s
}
```

3. 인자를 사용하지 않음 ➔ 함수 왼쪽)을 없앰.

```
myButton.setOnClickListener() {    it: View!
    val s:String = "안녕, 안드로이드 12!"
    textView.text = s
}
```

4. 함수의 인자가 하나 ➔ 괄호를 없앰

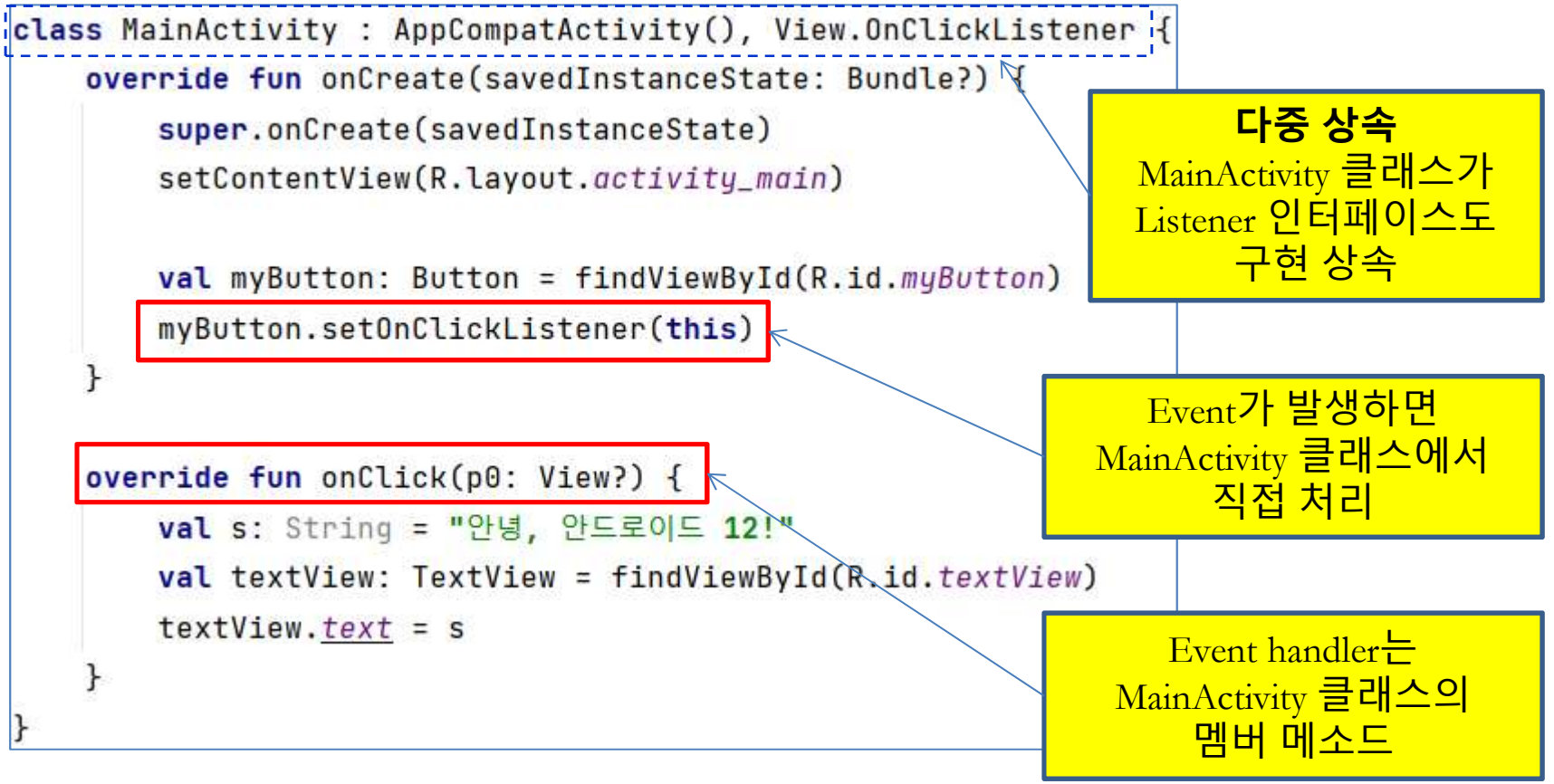
```
myButton.setOnClickListener {    it: View!
    val s:String = "안녕, 안드로이드 12!"
    textView.text = s
}
```

Quiz 3: 아래 코드는 왜 에러가 발생하지 않을까?

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

        val textView: TextView = findViewById(R.id.textView)
        val myButton:Button = findViewById(R.id.myButton)
        myButton.setOnClickListener {  it: View!
            val s: String = "안녕, 안드로이드 12!"
            textView.text = s
        }
    }
}
```

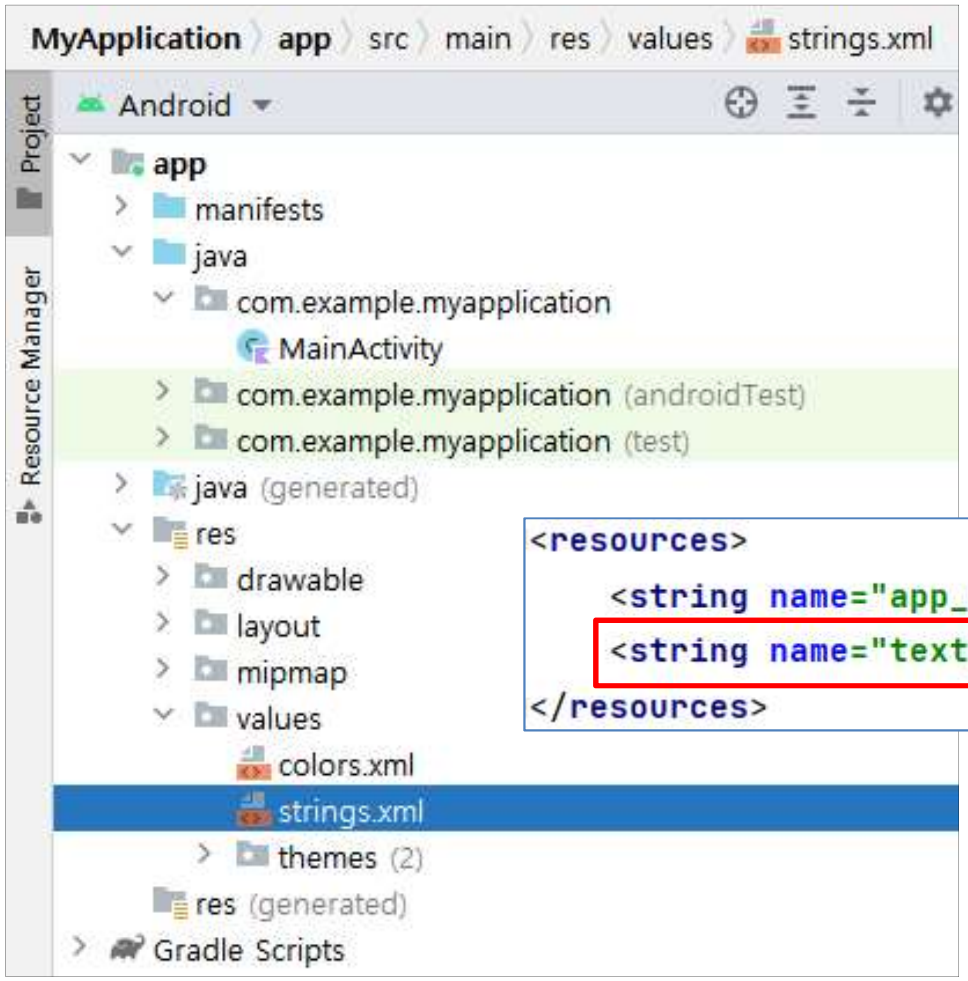
소스코드 – 7쪽

방법 3: Activity 클래스가 listener 인터페이스를 직접 상속



소스코드 – 8쪽

문자열 리소스 참조



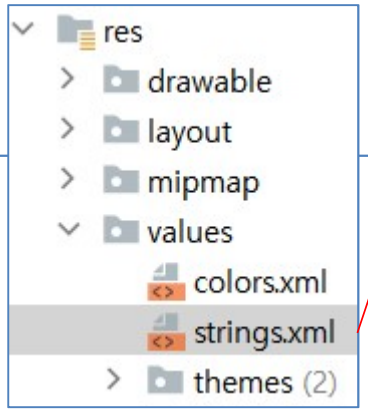
strings.xml

```
<resources>
    <string name="app_name">My Application</string>
    <string name="text_message">안녕, 안드로이드 12!</string>
</resources>
```

소스코드 – 9쪽

방법 2(람다 식)로 구현

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

        val textView: TextView = findViewById(R.id.textView)
        val myButton: Button = findViewById(R.id.myButton)
        myButton.setOnClickListener { it: View!
            val s = getString(R.string.text_message)
            textView.text = s
        }
    }
}
```




```
<string name="text_message">안녕, 안드로이드 12!</string>
```

소스코드 – 9쪽

스타일을 지정한 문자열 출력

strings.xml

```
<resources>
    <string name="app_name">My Application</string>
    <string name="text_message">안녕, <b>안드로이드 12!</b></string>
</resources>
```

MainActivity.kt

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

        val textView: TextView = findViewById(R.id.textView)
        val myButton: Button = findViewById(R.id.myButton)
        myButton.setOnClickListener {  it: View!
            val s :CharSequence = getText(R.string.text_message)
            textView.text = s
        }
    }
}
```

안녕, 안드로이드 12!

소스코드 – 10쪽

What to do next?

- 이벤트 핸들러를 구현하는 3가지 방법
  - 중첩 클래스
  - Anonymous class
  - Activity 클래스에서 직접 상속
  - **실습 1: 다양한 단위 변환 방식 적용**
  - 실습 2: 한 개 view에 대한 2개의 이벤트 처리
- 이벤트 종류
  - 실습 3: single touch 이벤트
  - 실습 4: multi-touch 이벤트

화면 밀도 (screen density)

화면 밀도 (screen density)

- 화면 면적 당 픽셀 수, 대개 1인치 당 픽셀 수를 나타내는 dpi(dots per inch)를 사용함
- 안드로이드의 6가지 화면밀도 분류와 dpi수

화면밀도 분류	dpi
ldpi(low)	120
mdpi(medium)	160
hdpi(high)	240
xhdpi(extra-high)	320
xxhdpi(extra-extra-high)	480
xxxhdpi(extra-extra-extra-high)	640

px	pixel(picture element)	디스플레이 기본 단위
dp(dip)	**density**-independent pixel	160dpi 밀도를 기준으로 한 픽셀.
sp(sip)	**scale**-independent pixel	글꼴 크기에 따라 픽셀 수가 달라짐. 텍스트 크기를 지정할 때 사용.

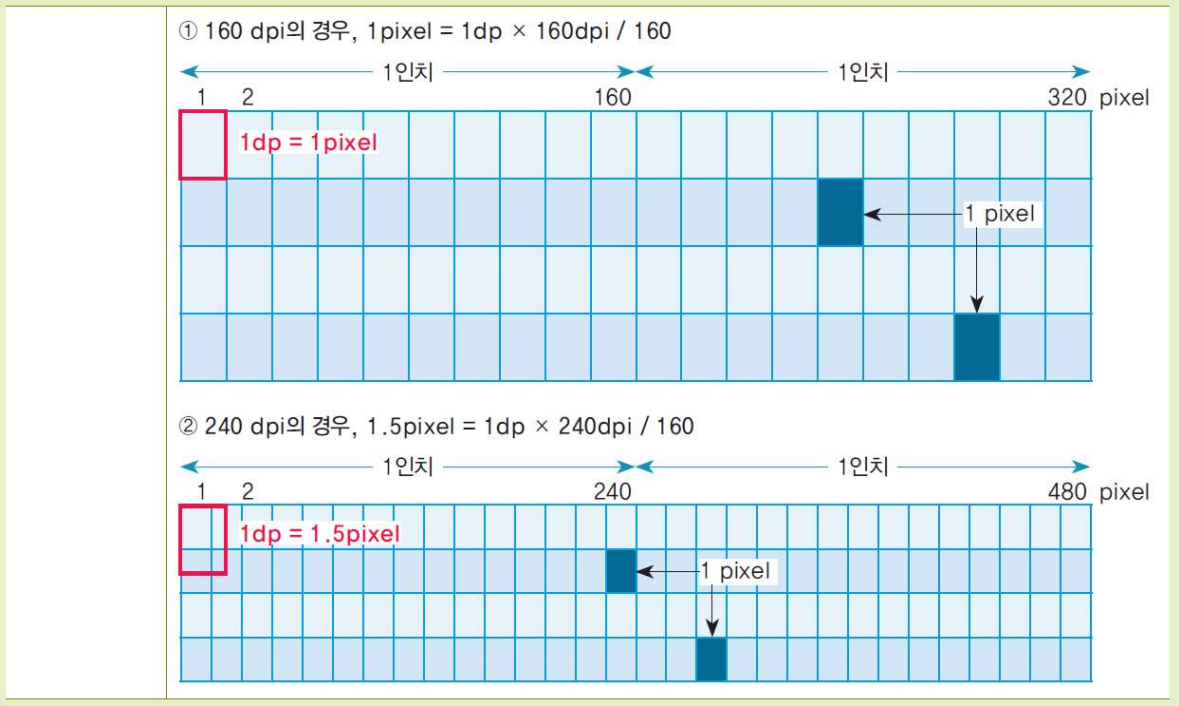
참고 http://angrytools.com/android/pixelcalc - Android pixel calculator

dp 와 px 관계

- dp : screen density와 무관한 가상 pixel

  px = dp x dpi / 160

  - Screen density가 **medium**(=160dpi) ➔ 1dp = **1**px
  - Screen density가 **high**(=240dpi) ➔ 1dp = **1.5**px



단위 변환 : dp → px

```
val density: Float = resources.displayMetrics.density
val densityDpi = resources.displayMetrics.densityDpi
Log.i(TAG,  msg: "density = $density") // screen density
Log.i(TAG,  msg: "densityDpi = $densityDpi") // dpi:dot per inch

var scale: Float = densityDpi/160F
Log.i(TAG,  msg: "scale = $scale") // scale

val dp = 16
val px = (scale*dp + 0.5).toInt()
Log.i(TAG,  msg: "dp = $dp, px = $px")
```

AVD의 dpi = 420dpi

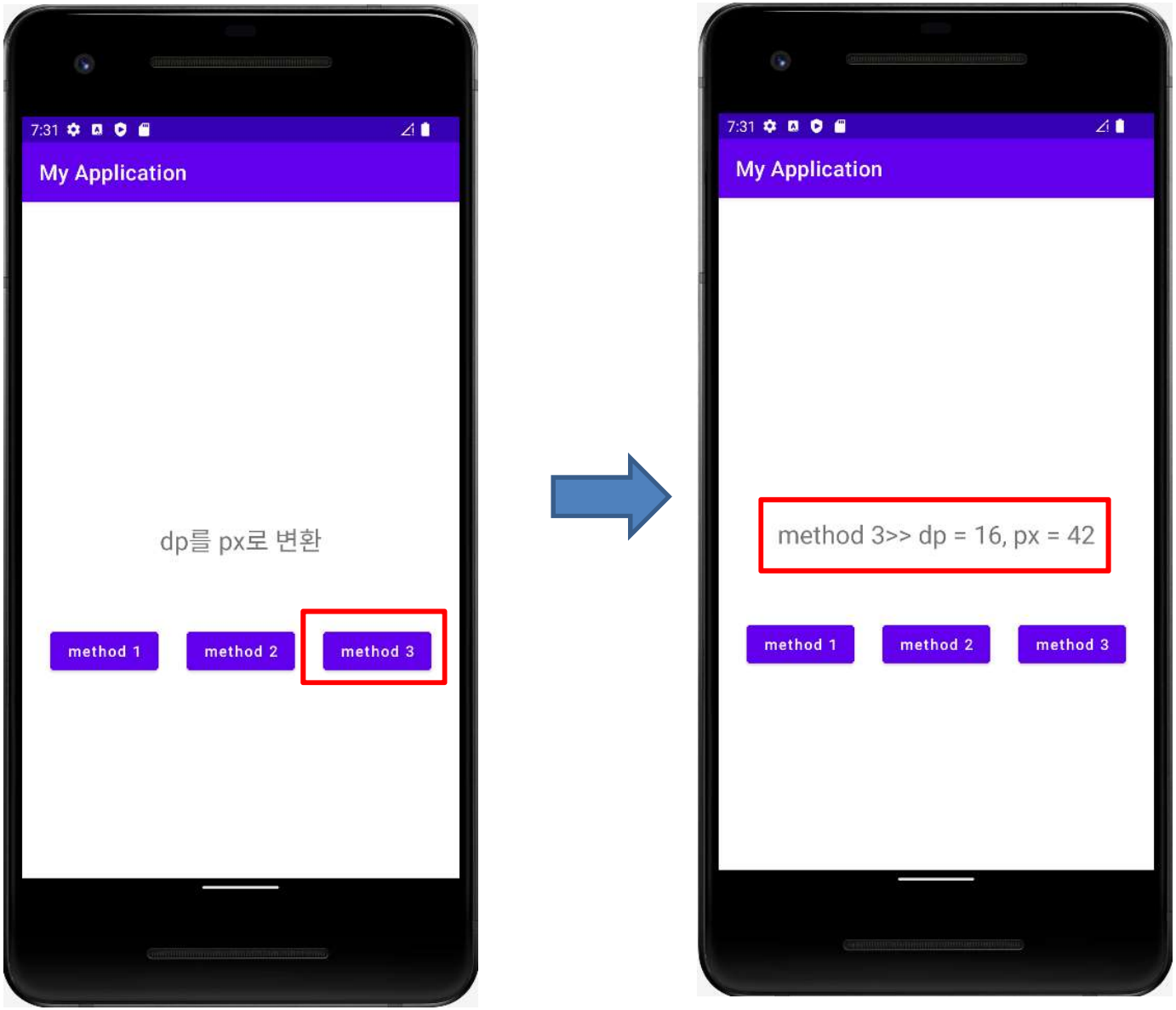
정수 변환 과정에서 반올림을 위해 0.5를 더함

0.4+0.5 =0.9 → 0
0.6+0.5 =1.1 → 1

LogCat 창에서 출력을 확인하려면 먼저 실행을 시켜야 함

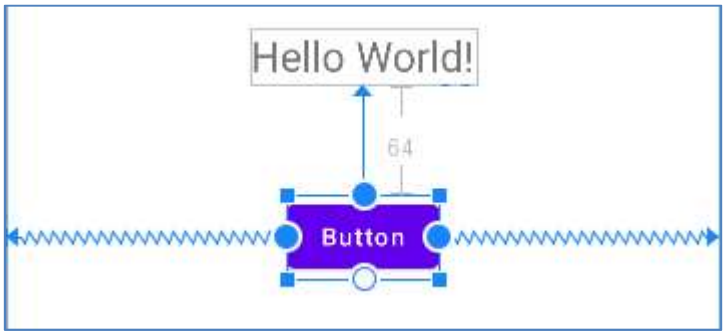
소스코드 – 11쪽

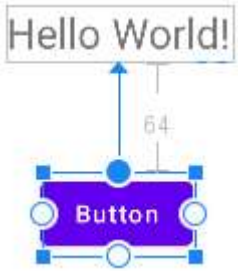
실습 1: 다양한 단위 변환 방법 적용

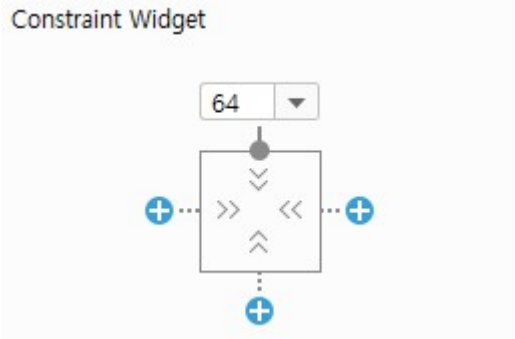


Layout – Horizontal chain으로 결합 (1/2)

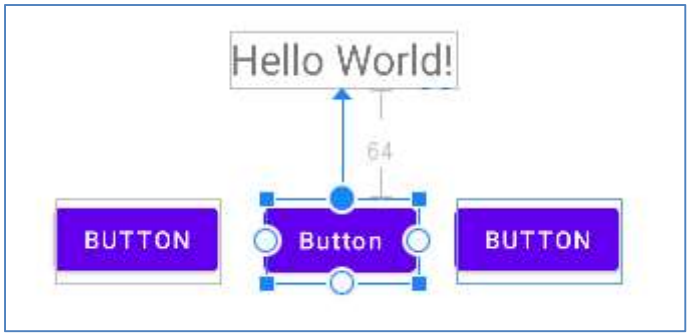
1. Button의 left 및 right constraint을 삭제



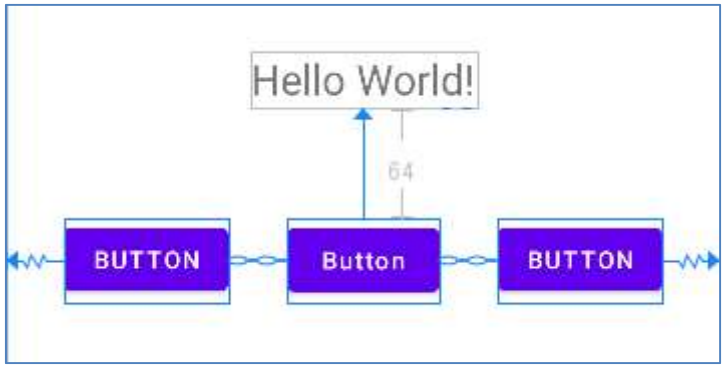




2. 왼쪽과 오른쪽에 Button 배치



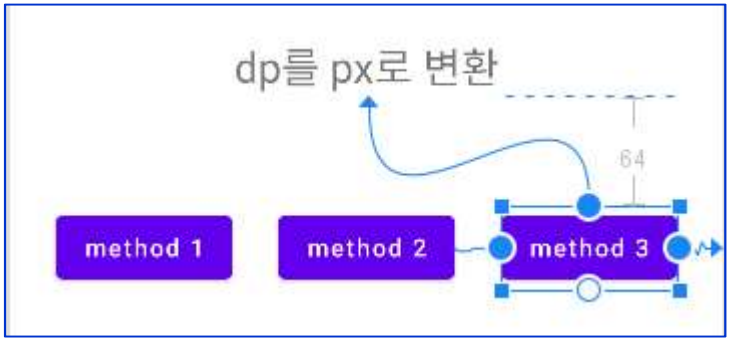
3. Button 3개를 Horizontal chain으로 연결



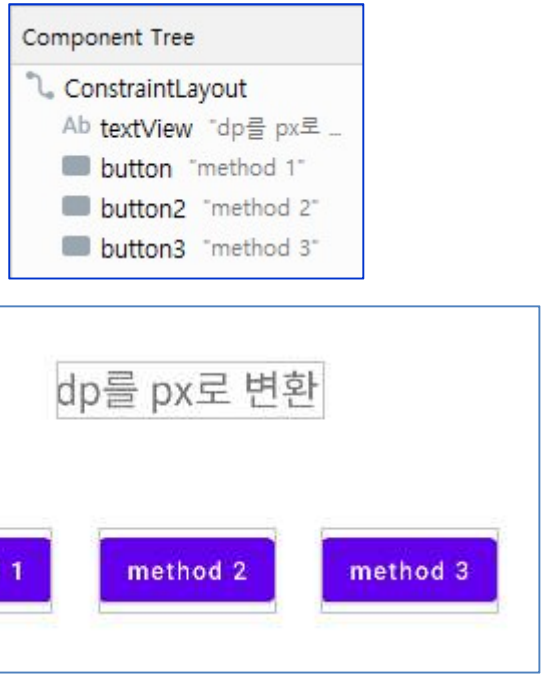
소스코드 – 12~13쪽

Layout – Horizontal chain으로 결합 (2/2)

4. Button 속성 및 TextView 속성 설정



5. **method 1** 버튼과 **method 3** 버튼의 top constraint을 TextView의 bottom constraint에 연결 (margin : 64dp)



소스코드 – 12~13쪽

실습 1: Activity

Method 1은
슬라이드 6

반올림 함수
적용

applyDimension
메소드는
Float 타입을
반환하기 때문에
정수 변환이 필요

매개변수
타입도 Float

```
val textView: TextView = findViewById(R.id.textView)
val dp = 16
val scale: Float = resources.displayMetrics.density

val button: Button = findViewById(R.id.button)
button.setOnClickListener {  it: View!
    val px:Int = (scale * dp + 0.5).toInt() // method 1
    val msg = "method 1>> dp = $dp, px = $px"
    textView.text = msg
}

val button2:Button = findViewById(R.id.button2)
button2.setOnClickListener {  it: View!
    val px:Int = (scale * dp).roundToInt() // method 2
    val msg = "method 2>> dp = $dp, px = $px"
    textView.text = msg
}

val button3:Button = findViewById(R.id.button3)
button3.setOnClickListener {  it: View!
    val px:Int = TypedValue.applyDimension( // method 3
        TypedValue.COMPLEX_UNIT_DIP, dp.toFloat(),
        resources.displayMetrics).toInt()
    val msg = "method 3>> dp = $dp, px = $px"
    textView.text = msg
}
```

소스코드 – 14쪽

What to do next?

- 이벤트 핸들러를 구현하는 3가지 방법
  - 중첩 클래스
  - Anonymous class
  - Activity 클래스에서 직접 상속
  - 실습 1: 다양한 단위 변환 방식 적용
  - **실습 2: 한 개 view에 대한 2개의 이벤트 처리**

- 이벤트 종류
  - 실습 3: single touch 이벤트
  - 실습 4: multi-touch 이벤트

Listener와 callback method

Listener	Callback method	Description
View. OnClickListener	**onClick( )**	View를 touch(click). 또는 내비게이션 key나 track ball로 이동한 후 view를 선택할 때 호출
View. OnLongClickListener	**onLongClick( )**	일정 시간 이상 움직이지 않고 view를 계속 touch(click)할 때 호출
View. OnFocusChangeListener	**onFocusChange( )**	Focus를 갖고 있는 view에서 다른 view로 focus를 전환할 때 호출
View. OnKeyListener	**onKey( )**	View가 focus를 갖고 있을 때 soft key를 입력하면 호출
View. OnTouchListener	**onTouch( )**	Touch event 동작을 했을 때 호출

실습 2: 한 개 view에 대한 2개 이벤트 처리

실습 2: Activity

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

        var myButton: Button = findViewById(R.id.myButton)
        var textView: TextView = findViewById(R.id.textView)
        myButton.setOnClickListener (object : View.OnClickListener {
            override fun onClick(v: View?) {
                textView.text = "Button clicked"
            }
        })

        myButton.setOnLongClickListener(object : View.OnLongClickListener {
            override fun onLongClick(v: View?): Boolean {
                textView.text = "Long button click"
                return false
            }
        })
    }
}
```

이벤트를 제대로 처리 못했으니
등록된 다른 listener가 처리하라고 알림.

소스코드 – 15쪽

람다 식 변환


var myButton: Button = findViewById(R.id.myButton)
var textView: TextView = findViewById(R.id.textView)
myButton.setOnClickListener (object : View.OnClickListener {
Convert to lambda
Import members from 'android.view.View'
Convert to with
Introduce import alias
Convert to run
Add full qualifier
Press Ctrl+Shift+I to open preview
on clicked"
er(object : View.OnLongClickListener {
v: View?): Boolean {
textView.text = "Long button click"
return false
}

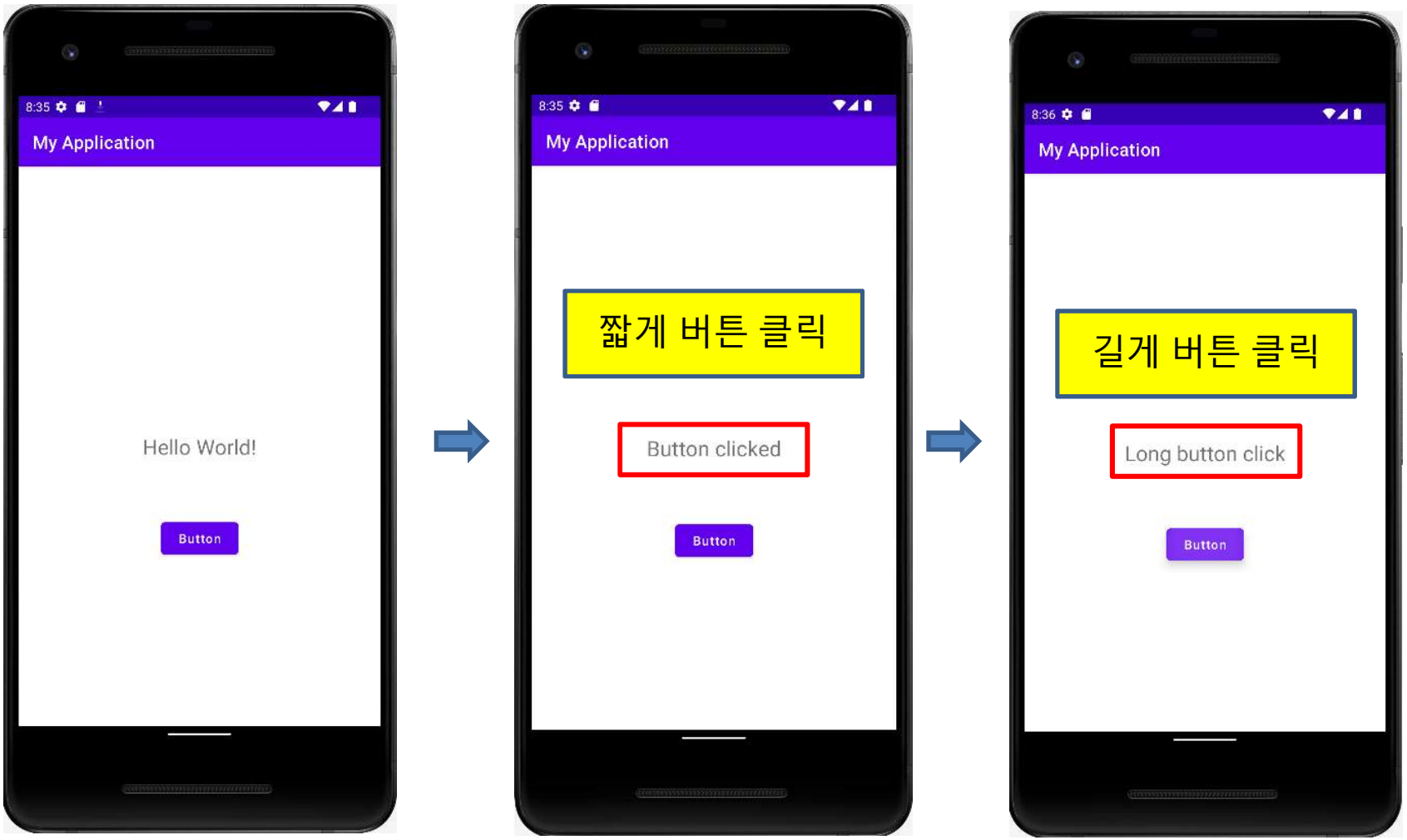
실습 2: Activity (람다 식)

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

        var myButton: Button = findViewById(R.id.myButton)
        var textView: TextView = findViewById(R.id.textView)
        myButton.setOnClickListener {  it: View!
            textView.text = "Button clicked"
        }

        myButton.setOnLongClickListener {  it: View!
            textView.text = "Long button click"
            false  ^setOnLongClickListener
        }
    }
}
```

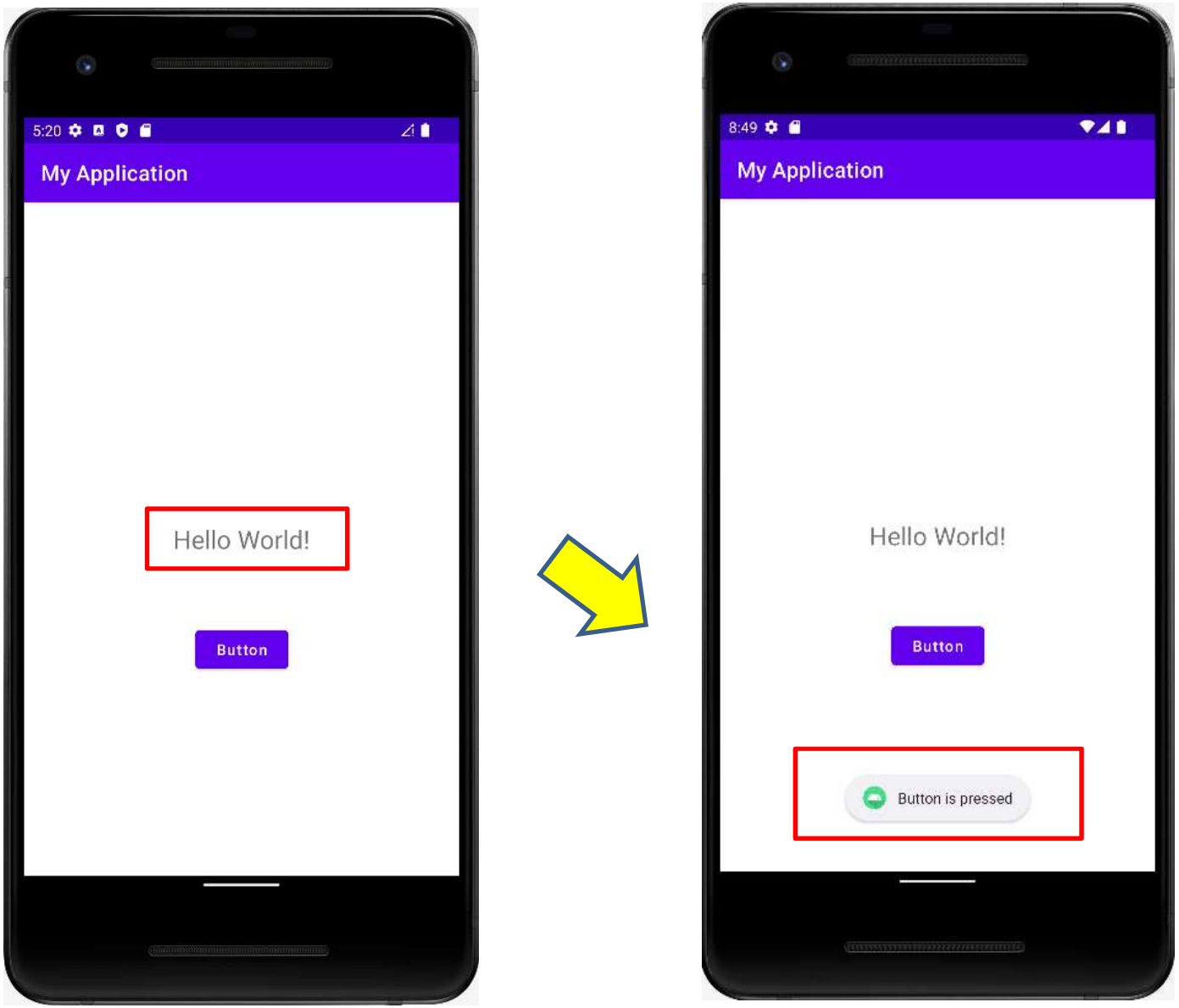
gray font로 표시된 내용은 Android Studio에서 제공하는 hint message
➔ 입력하면 안됨.

소스코드 – 16쪽

Quiz 4: onLongClick()의 return 값을 true로 바꾸면 결과는 어떻게 달라지나요?



버튼을 클릭하면 Toast 창에 메시지 출력



방법 2(람다 식)로 구현

```
var myButton: Button = findViewById(R.id.myButton)
myButton.setOnClickListener {  it: View!
    Toast.makeText(
        applicationContext, text: "Button is pressed",Toast.LENGTH_SHORT).show()
}

myButton.setOnLongClickListener {  it: View!
    Toast.makeText(
        applicationContext, text: "Long Button click", Toast.LENGTH_SHORT).show()
    false  ^setOnLongClickListener
}
```

this@MainActivity

this 대신 **this@MainActivity** 와 같이
UI를 구현하는 클래스 이름을 정확하게 적어야 함.

그러나, ***applicationContext*** 를 사용하는 것이 좋음.
➔ 항상 UI 클래스(Activity)를 찾아주기 때문

소스코드 – 17쪽

Toast 창

```
Toast.makeText(applicationContext,
    "Button is pressed.",
    Toast.LENGTH_LONG).show()
```

Toast 는 **화면**에 메시지(문자열)를 잠깐 출력하고 사라짐
applicationContext ➔ 화면
"Button is pressed." ➔ 출력 문자열
Toast.LENGTH_SHORT, Toast.LENGTH_LONG ➔ 출력 시간 설정

applicationContext
➔ Activity 컴포넌트를 구현하고 있는 객체.
➔ App. 실행이 끝날 때까지 같은 객체를 참조.

context = 화면 = Activity 컴포넌트를 구현하고 있는 객체.

Quiz 5: 버튼을 클릭했을 때 Toast 창에도 메시지를 띄우고, TextView에도 문자열을 출력하도록 코드를 수정하세요.