
Kotlin을 이용한 Android 프로그래밍

가속도 센서를 이용한 수평 측정기 만들기

Contents

- I. 액티비티 생명주기
- II. 센서를 이용한 수평 맞추기 앱

액티비티 생명주기

▶ 컴포넌트(Component)

- ▶ 시스템에서 재사용을 위한 부품의 의미로 시작된 개념
- ▶ 소프트웨어 컴포넌트는 하드웨어의 그래픽카드와 같은 개념으로 **독립적인 기능을 수행**하는 소프트웨어 모듈
 - ▷ 독립적으로 실행 및 교체, 수정, 개선 될 수 있어야 함

▶ 안드로이드 4대 컴포넌트

- ▶ 액티비티, 서비스, 브로드캐스트 리시버, 콘텐츠 프로바이더
- ▶ 각각의 컴포넌트는 애플리케이션이 설치되면 안드로이드 시스템에서 이들에 대한 정보를 요구
- ▶ AndroidManifest.xml에 해당 정보가 들어있음

액티비티 생명주기

▶ 안드로이드 4대 컴포넌트

▶ 액티비티(Activity) :

▷ 화면을 구성하는 가장 기본적인 컴포넌트

▶ 서비스(Service) :

▷ 액티비티와 상관없이 백그라운드에서 동작하는 컴포넌트(ex:백신 등)

▶ 브로드캐스트 리시버(Broadcast Receiver) :

▷ 문자 메시지 도착, 배터리 방전, 네트워크 환경 변화 등이 발생하면 모든 어플에게 방송 신호가 보내지는데 이를 감지하여 반응하는 컴포넌트

▶ 콘텐츠 프로바이더(Content Provider)

▷ 어플리케이션 사이에 데이터를 상호 공유하기 위한 컴포넌트

액티비티 생명주기

▶ 액티비티(Activity)

- ▶ 안드로이드 폰에 나타나는 화면 하나하나를 지칭
- ▶ 일반적으로 액티비티 하나당 XML 파일 하나를 만들어서 사용
 - ▷ setContentView(activity_main.xml)
- ▶ MainActivity.kt 코드는 Activity 클래스를 상속받으므로 액티비티라고 부름
 - ▷ AppCompatActivity 클래스가 Activity 클래스로부터 상속 받았음

액티비티 생명주기

▶ 액티비티 생명주기

▶ 액티비티의 생성부터 소멸까지의 주기를 뜻함

▷ 액티비티의 상태정보가 변화하는 것

▶ 안드로이드에서는 실행되는 앱의 상태를 시스템에서 직접관리

▷ 처음 실행되어 메모리에 생성되는 과정부터 실행과 중지, 메모리에서 해제되는 여러 과정을 상태 정보로 가지고 있게 되고 이러한 상태 정보는 시스템에서 각각의 상태에 해당하는 메소드를 자동으로 호출

▶ 안드로이드 응용프로그램은 화면이 작아 동시에 여러 개의 액티비티(화면)가 나올 수 없음

▷ 앞에 나오는 화면 하나만 활성화된 상태이고 나머지는 모두 비활성화된 상태로 남게 됨

액티비티 생명주기

▶ 액티비티 실행 상태

상 태	설 명
실행(Running)	화면 상에 액티비티가 보이면서 실행되어 있는 상태. 액티비티 스택의 최상위에 있으며 포커스를 가지고 있음
일시 중지(Paused)	사용자에게 보이기는 하지만 다른 액티비티가 위에 있어 포커스를 받지 못하는 상태. 대화상자가 위에 있어 일부가 가려져 있는 경우에 해당함
중지(Stopped)	다른 액티비티에 의해 완전히 가려져 보이지 않는 상태

액티비티 생명주기

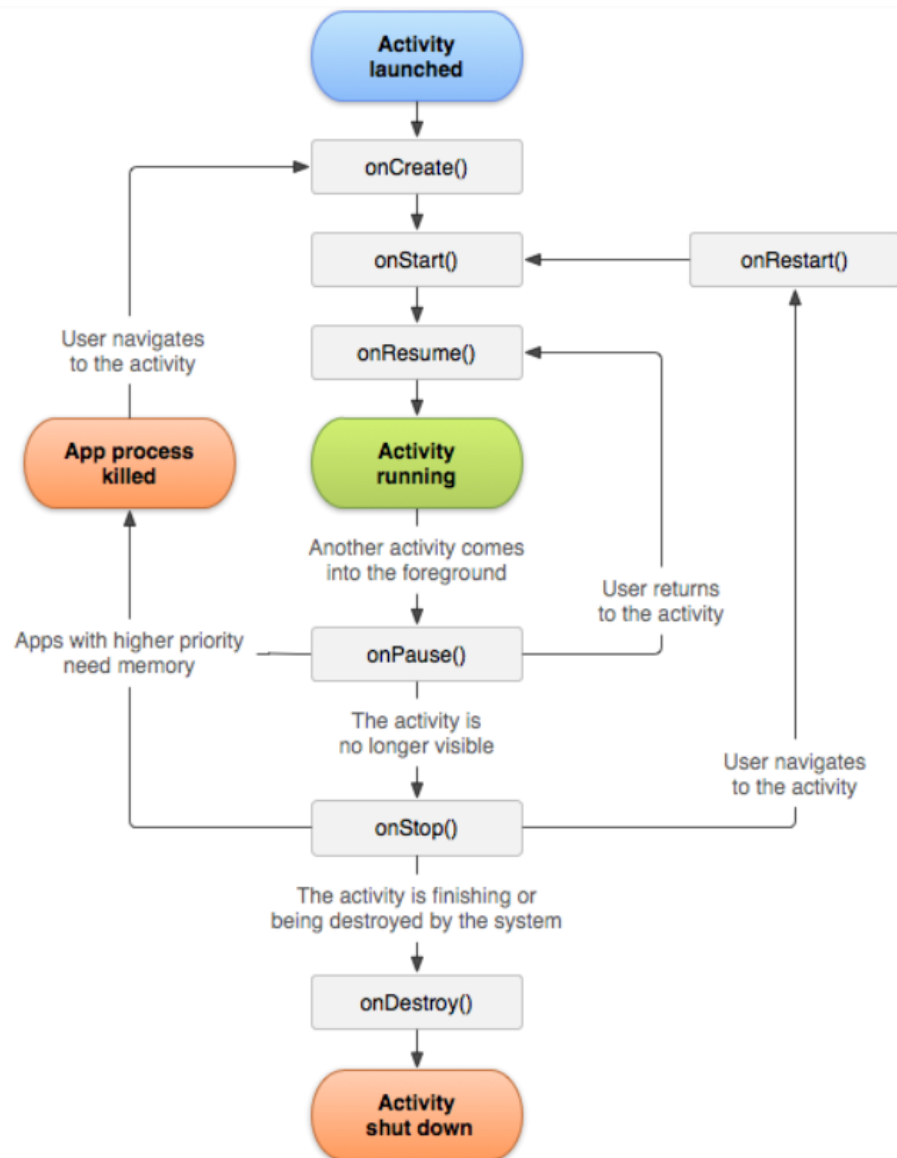
▶ 액티비티의 생명주기(LifeCycle)

▶ 액티비티는 특정 시점에 호출되는 다양한 메소드가 존재

▷ onCreate()메소드는 생성 시점에 호출

▶ 특정 타이밍에 호출되는 메소드를 콜백(callback) 메서드라고 함

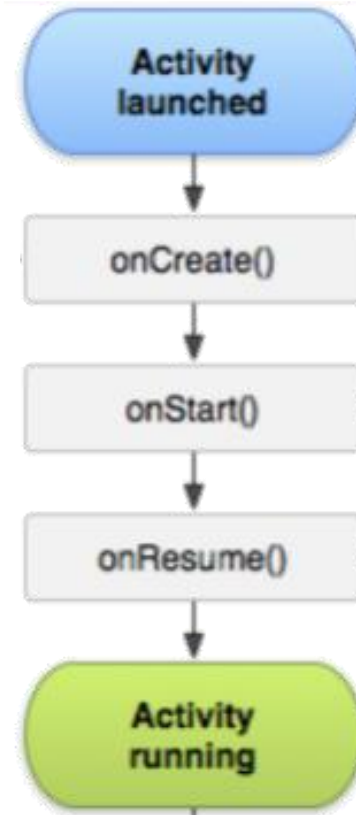
▶ 화면이 비활성화인 상태에서도 센서가 동작하면 배터리가 빨리 소모되므로 생명주기를 숙지하여 적절한 타이밍에 사용



액티비티 생명주기

▶ 액티비티 시작

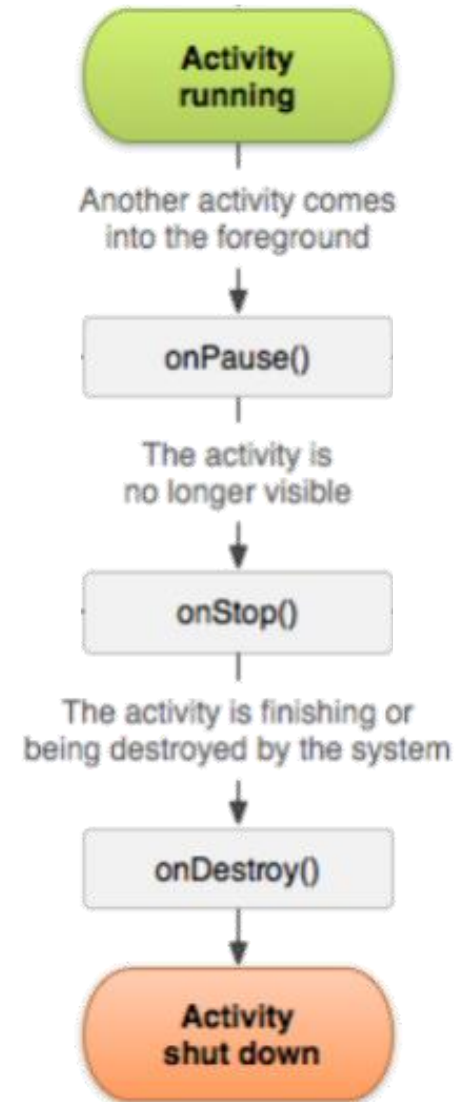
- ▶ 액티비티가 시작되면 가장 먼저 onCreate()메서드가 호출
- ▶ onCreate()메서드를 오버라이드하여 프로그램을 작성하면 액티비티가 시작되면서 작성한 프로그램이 자동으로 시작
 - ▶ 프로젝트를 생성하면 onCreate()메서드는 자동으로 보임
- ▶ 이후에는 onStart()메서드와 onResume()메서드 순으로 호출



액티비티 생명주기

▶ 액티비티 종료

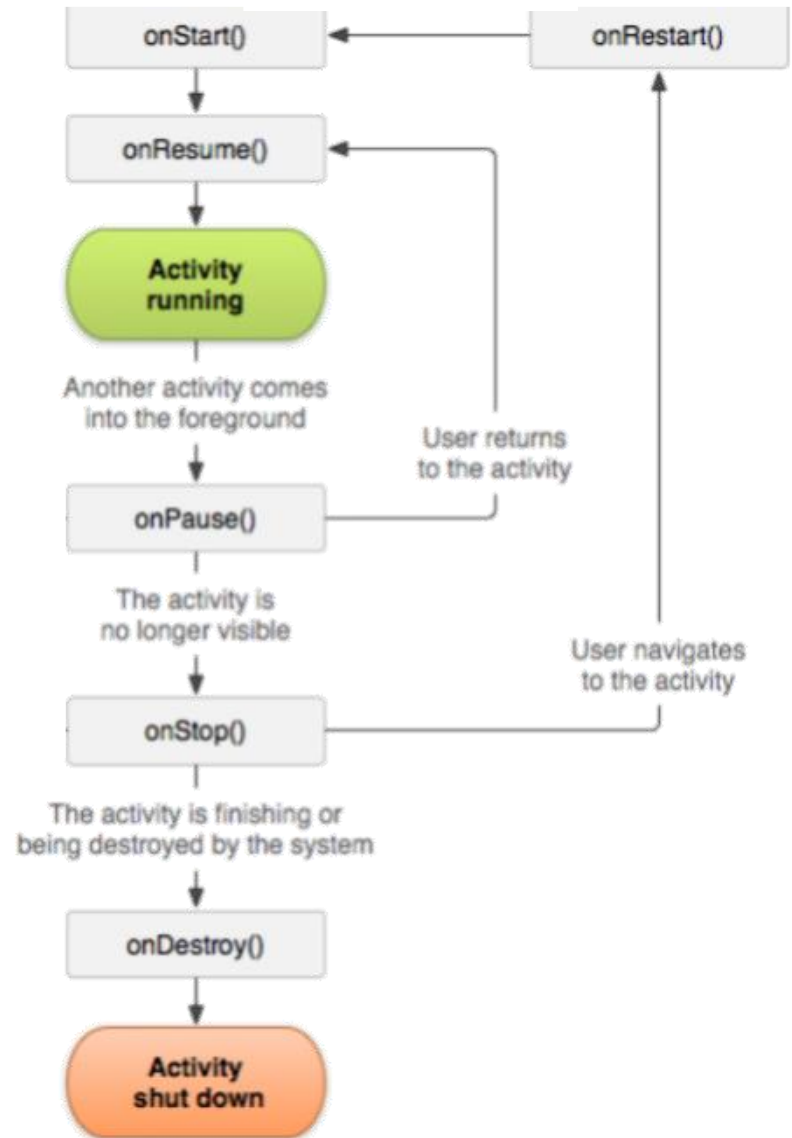
- ▶ 액티비티가 종료될 때는 화면에 보이지 않게 되는 순간 제일 먼저 onPause()메서드가 호출되고 완전히 보이지 않게 되면 onStop()메서드가 호출된 후 onDestroy()메서드가 호출됨



액티비티 생명주기

▶ 액티비티 재개

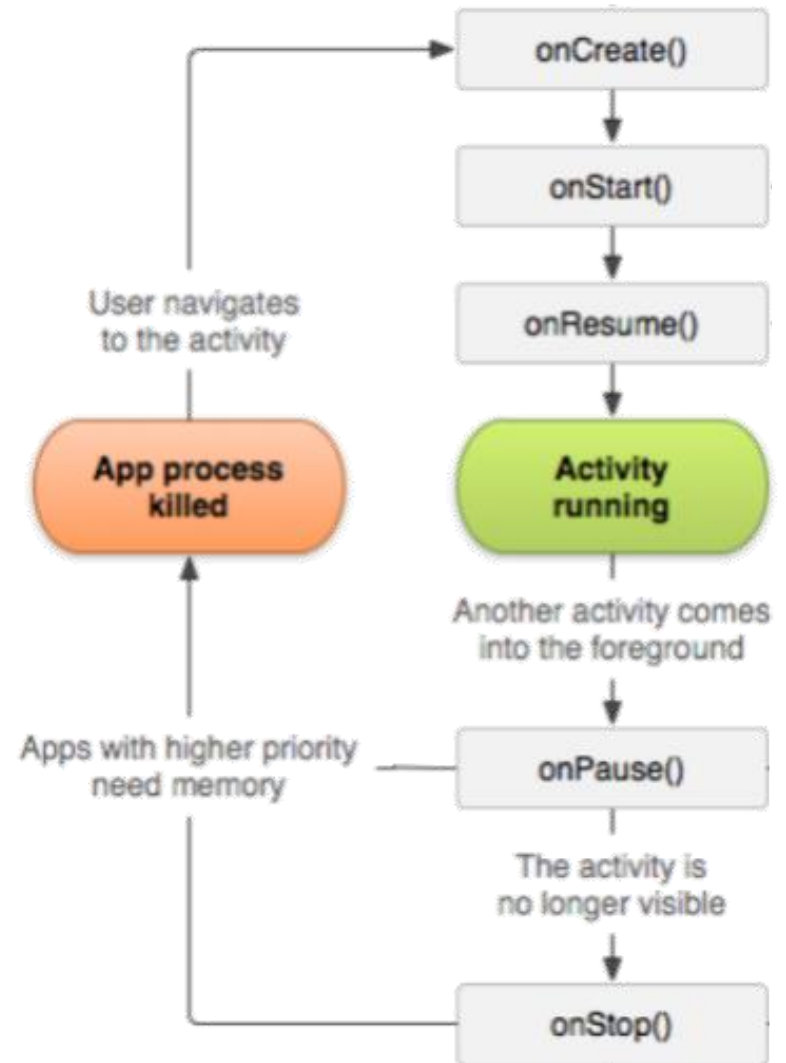
- ▶ 앱이 실행 중일 때 액티비티가 종료되지 않고 백그라운드에서 대기하는 경우가 있음
 - ▶ 다른 앱이 실행되거나 홈 키를 누르거나 화면이 꺼진 경우
 - ▶ onPause() 나 onStop()까지만 호출되고 대기
- ▶ 이때 다시 액티비티가 전면에 나오는 경우에는 onRestart() - onStart() - onResume() 순으로 호출됨



액티비티 생명주기

▶ 프로세스 강제 종료

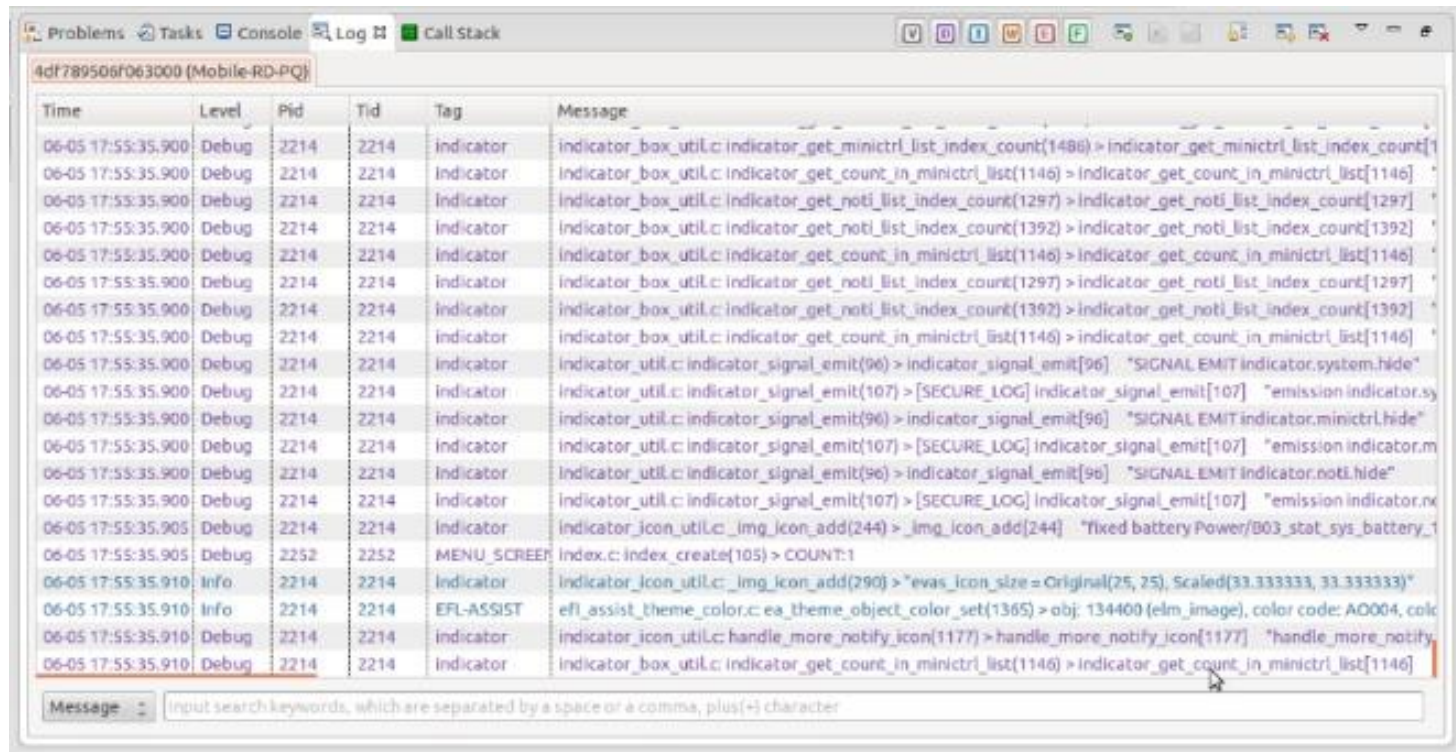
- ▶ 안드로이드의 모든 앱은 백그라운드 상태에서 메모리 부족 등으로 강제 종료될 수 있음
 - ▷ 액티비티도 마찬가지
- ▶ 이때 다시 앱이 실행되면 onCreate()메서드부터 호출됨



액티비티 생명주기

▶ 로그(Log)

- ▶ 로그는 시스템이나 응용 프로그램의 동작에 대한 상세한 기록
- ▶ 로그는 디버깅과 다르게 중단점을 걸지 않아도 항상 출력되며 실시간으로 프로그램의 상태를 보여 준다는 점에서 유용



액티비티 생명주기

▶ 로그캣(LogCat)

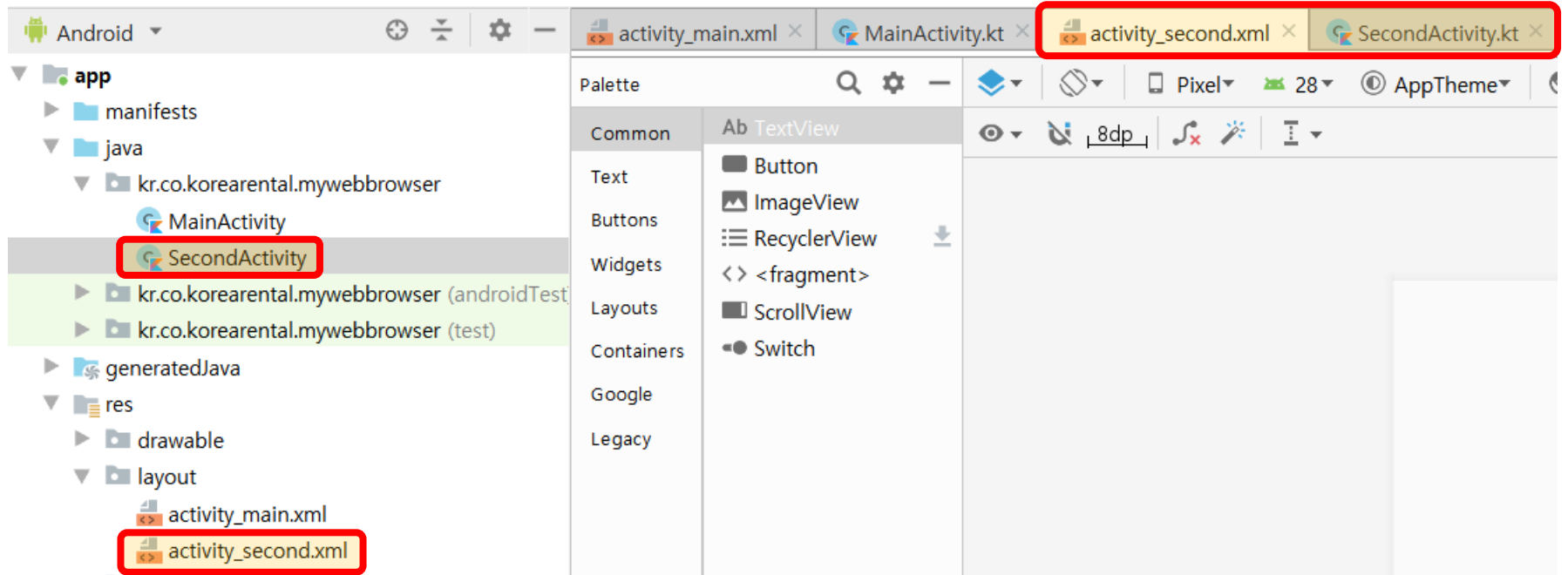
- ▶ 오류의 원인을 파악하는 방법 중에 하나가 로그를 남기는 것
- ▶ 안드로이드 스튜디오는 로그캣 화면을 통하여 로그를 확인할 수 있음
- ▶ 첫번째 인수는 로그를 출력하는 주체의 이름이며 차후 필터링 됨
- ▶ 두번째 인수가 로그의 실제 내용인데 단순 사실을 기록할 수도 있고 + 연산자로 연결하여 변수 값을 출력 가능

	설명
Log.d("태그", "메시지")	Debugging : 디버깅 용도로 남기는 로그
Log.e("태그", "메시지")	Error : 가장 심각한 오류 발생시 남기는 로그
Log.i("태그", "메시지")	Information : 정보를 남기기 위한 로그
Log.v("태그", "메시지")	Verbose : 상세한 기록을 남기기 위한 로그
Log.w("태그", "메시지")	Warning : 경고 수준을 남기기 위한 로그

액티비티 생명주기

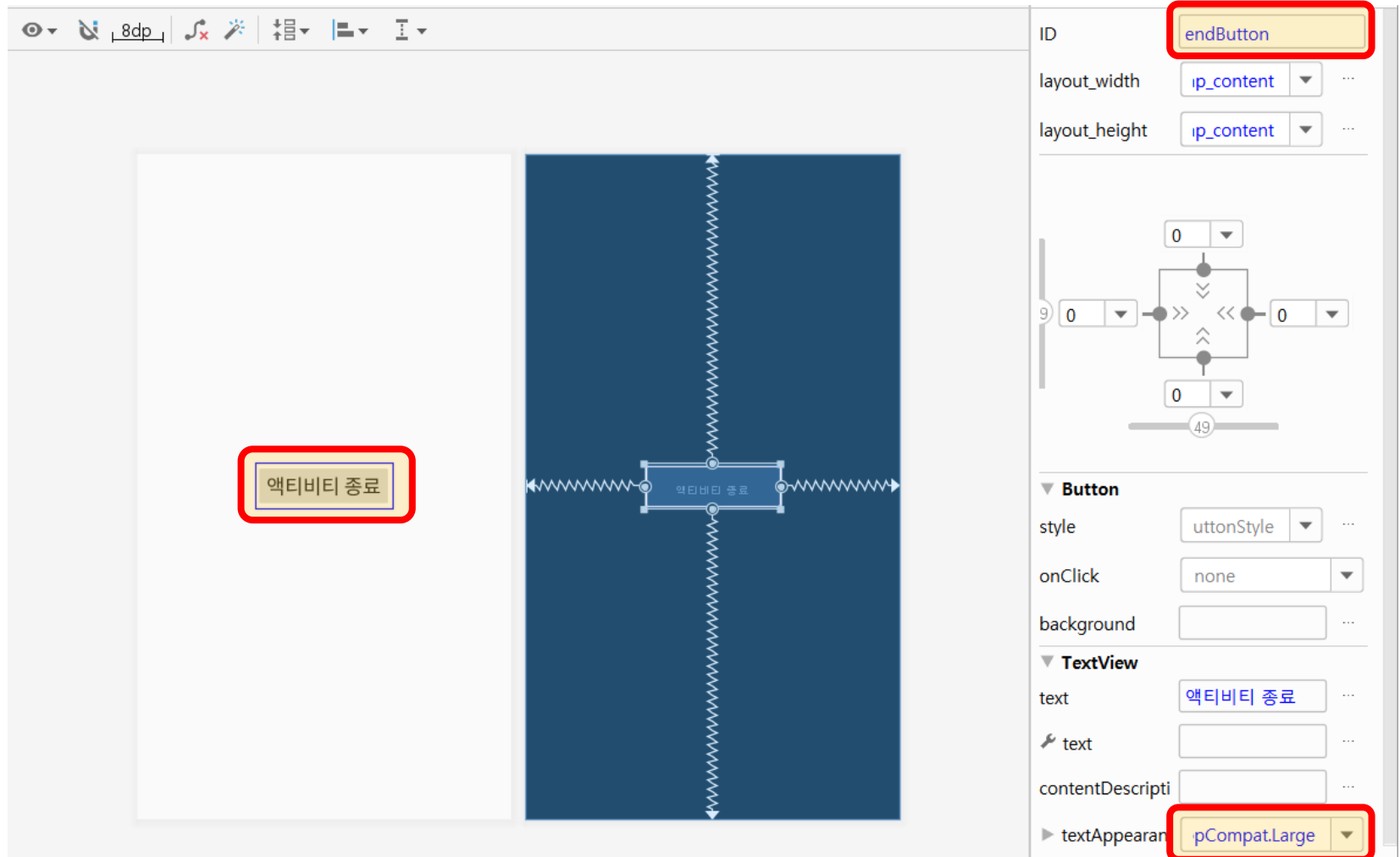
▶ 앞서 진행한 웹 브라우저 예제에 액티비티 추가

▶ SecondActivity



액티비티 생명주기

▶ activity_second.xml에 버튼 추가



액티비티 생명주기

▶ 업 네비게이션 설정

▶ android:parentActivityName=".MainActivity"

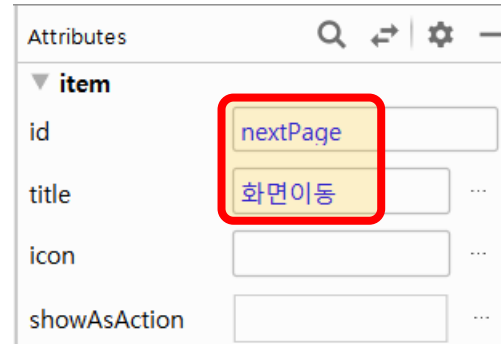
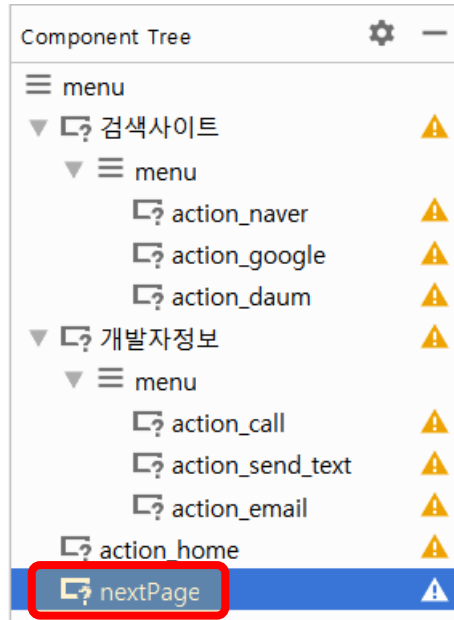
```
<activity  
    android:name=".ResultActivity"  
    android:parentActivityName=".MainActivity">  
</activity>
```

▶ 버튼 클릭 시 액티비티가 종료되도록 코드 작성

```
endButton.setOnClickListener { it: View!  
    finish()  
}
```

액티비티 생명주기

▶ 옵션 메뉴에 화면전환 메뉴 추가



액티비티 생명주기

▶ onOptionsItemSelected 메소드에 화면 전환 코드 추가

```
R.id.action_email ->{  
    email(email: "text@example.com", subject: "좋은 사이트", webView.url)  
    return true  
}  
R.id.nextPage ->{  
    startActivity<SecondActivity>()  
    return true  
}
```

액티비티 생명주기

▶ MainActivity에 액티비티 생명주기 상에서 콜백되는 메소드 오버라이드 후 로그 추가

▶ onStart(), onResume(), onPause(), onStop(), onDestroy(), onRestart()

▶ tag : LifecycleTest

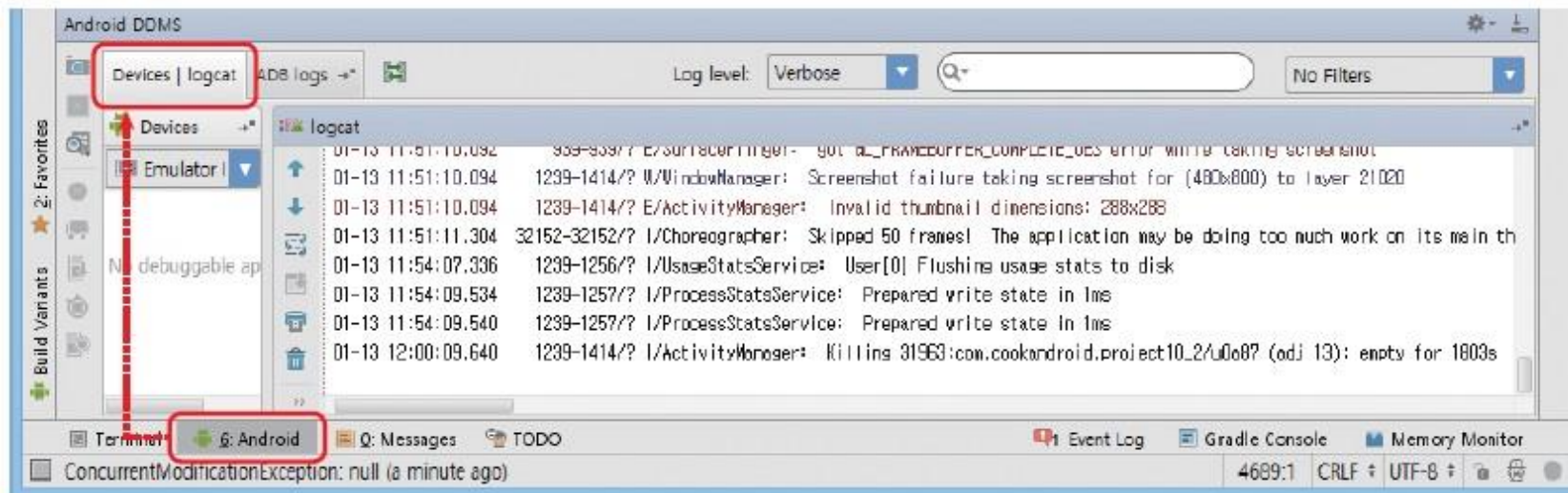
▶ msg : 메소드 이름

```
override fun onStart() {  
    super.onStart()  
  
    Log.i( tag: "LifecycleTest", msg: onStart()_Main )  
}
```

액티비티 생명주기

▶ 프로젝트 실행 및 결과 확인

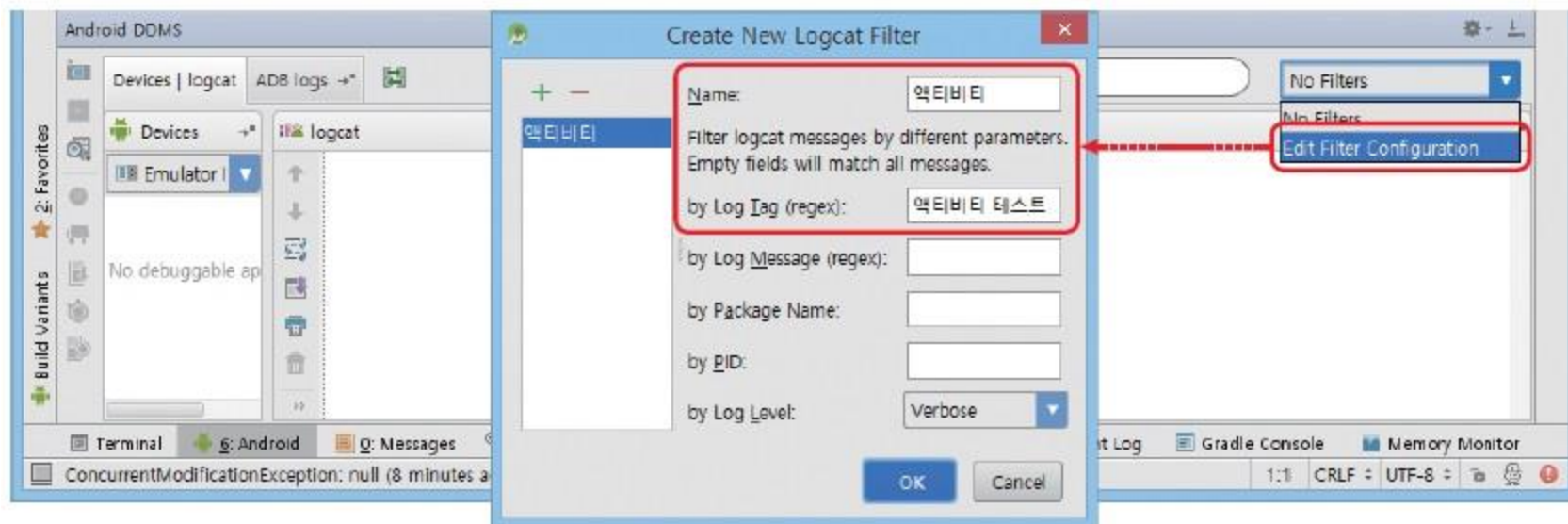
- ▶ 로그캣 화면이 보이지 않으면 Android Studio 아래쪽 탭 중에서 'Android'를 클릭하고 'Device | logcat' 탭을 선택



액티비티 생명주기

▶ 프로젝트 실행 및 결과 확인

▶ 로그필터 등록



액티비티 생명주기

▶ 프로젝트 실행 및 결과 확인

▶ 응용프로그램을 실행하여 로그켓을 확인

▶ 만약 로그켓에 내용이 보이지 않으면 실행 단계마다 왼쪽 '액티비티' 필터를 클릭



센서를 이용한 수평 측정기

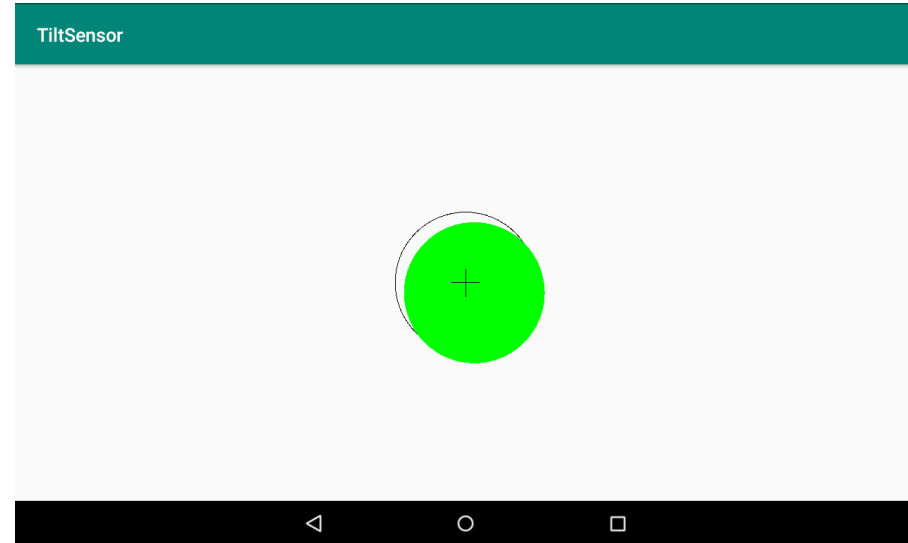
▶ 프로젝트 명 : TiltSensor

▶ 기능

- ▶ 기기를 기울이면 수평을 측정할 수 있음
- ▶ 화면에 표시된 원이 가운데로 이동하면 수평

▶ 구성요소

- ▶ SensorManager : 센서 관리자
- ▶ SensorEventListener : 센서 이벤트를 수신하는 리스너
- ▶ 커스텀 뷰 : 개발자가 직접 만드는 뷰



센서를 이용한 수평 측정기

▶ 프로젝트 설계

- ▶ 가속도 센서를 사용하여 수평 측정기를 작성
- ▶ 가로 모드로 고정하고 사용 중에는 화면이 꺼지지 않도록 설정
- ▶ 수평 측정기 화면은 커스텀 뷰로 작성하고 그래픽 관련 API를 사용하여 그림 완성

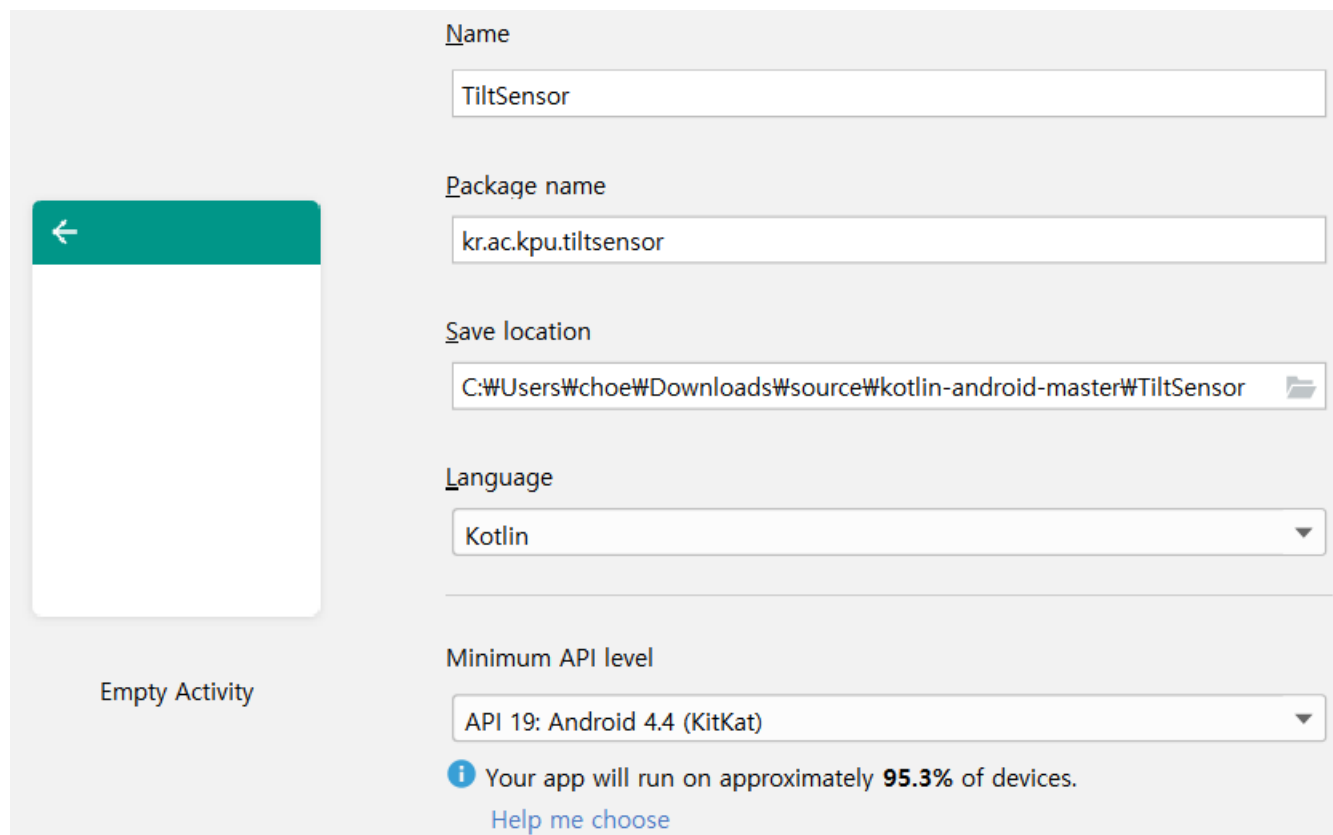
▶ 구현 순서

- 1) 프로젝트 생성
- 2) 센서 사용
- 3) 커스텀 뷰 작성

센서를 이용한 수평 측정기

▶ 프로젝트 생성

- ▶ 프로젝트 명 : TiltSensor
- ▶ minSdkVersion : 19
- ▶ 기본 액티비티 : Empty Activity



Name
TiltSensor

Package name
kr.ac.kpu.tiltsensor

Save location
C:\Users\wchoe\Downloads\source\kotlin-android-master\TiltSensor

Language
Kotlin

Minimum API level
API 19: Android 4.4 (KitKat)

Empty Activity

Info Your app will run on approximately **95.3%** of devices.
[Help me choose](#)

액티비티 생명주기

▶ 앱 아이콘 변경하기

▶ 강의 사이트에서 kt_levelmeter.png파일 다운로드 후 메니페스트에 추가

```
<application
```

```
    android:allowBackup="true" // 기기에 연결된 구글 계정에 앱 데이터를 25MB까지 자동으로 백업
```

```
    android:icon="@mipmap/kr_levelmeter"
```

```
    android:label="TiltSensor"
```

```
    android:roundIcon="@mipmap/kr_levelmeter"
```

```
    android:supportRtl="true"
```

```
    android:theme="@style/AppTheme">
```

▶ supportsRtl = "true"

▶ 이슬람 문화권의 레이아웃 지원

▶ 안드로이드 17이상에서 지원

■ 4.2 JellyBean 이상



센서를 이용한 수평 측정기

▶ 센서 사용하기

- ▶ 센서 사용 방법은 대부분 유사하기 때문에 하나의 센서만 잘 사용하면 다른 센서들도 응용이 가능

▶ 센서(Sensor)

- ▶ 안드로이드 기기에는 다양한 센서가 내장되어 있음
- ▶ 안드로이드가 공식적으로 제공하는 센서 목록
- ▶ 아래 센서가 모두 내장된 것은 아니며 제조사에 따라 여기에 포함되지 않은 센서를 사용하기도 함

센서	설명	용도
TYPE_ACCELEROMETER	가속도 센서	동작 감지 (흔들림, 기울임 등)
TYPE_AMBIENT_TEMPERATURE	주변 온도 센서	대기 온도 모니터링
TYPE_GRAVITY	중력 센서	동작 감지 (흔들림, 기울임 등)
TYPE_GYROSCOPE	자이로 센서	회전 감지
TYPE_LIGHT	조도 센서	화면 밝기 제어
TYPE_LINEAR_ACCELERATION	선형 가속도 센서	단일 축을 따라 가속 모니터링
TYPE_MAGNETIC_FIELD	자기장 센서	나침반
TYPE_ORIENTATION	방향 센서	장치 위치 결정
TYPE_PRESSURE	기압 센서	공기압 모니터링 변화
TYPE_PROXIMITY	근접 센서	통화 중인지 검사
TYPE_RELATIVE_HUMIDITY	상대 습도 센서	이슬점, 상대 습도를 모니터링
TYPE_ROTATION_VECTOR	회전 센서	모션 감지, 회전 감지
TYPE_TEMPERATURE	온도 센서	장치의 온도 감지

센서를 이용한 수평 측정기

▶ 센서 사용과정

▶ SensorManager 인스턴스를 가져옴

▷ 센서를 사용하려면 안드로이드가 제공하는 센서 매니저 서비스 객체가 필요

▷ 센서 매니저는 안드로이드 기기의 각 센서 접근 및 리스너의 등록/취소, 이벤트를 수집하는 방법을 제공

▶ 센서 목록 중에 하나를 getDefaultSensor() 메서드에 지정하여 Sensor 객체를 가져옴

▶ onResume() 메서드에서 registerListener() 메서드로 센서의 감지를 등록

▶ onPause()메서드에서 unregisterListener() 메서드로 센서의 감지를 해제

센서를 이용한 수평 측정기

▶ 센서 매니저에 대한 참조

- ▶ 장치에 있는 센서를 사용하려면 가장 먼저 센서 매니저에 대한 참조를 얻어야 함
- ▶ getSystemService() 메서드에 SENSOR_SERVICE 상수를 전달하여 SensorManager 클래스의 인스턴스를 생성
- ▶ 지연된 초기화를 사용하여 sensorManager 변수를 처음 사용할 때 getSystemService() 메서드로 SensorManager 객체를 얻음

▶ as 키워드로 형변환

```
class MainActivity : AppCompatActivity() {
```

```
    private val sensorManager by lazy {  
        getSystemService(Context.SENSOR_SERVICE) as SensorManager  
    }
```

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

```
}
```

센서를 이용한 수평 측정기

▶ 센서 등록

- ▶ 액티비티가 동작할 때만 센서가 동작해야 배터리 불필요한 소모를 방지하기 때문에 센서의 사용 등록은 onResume()메서드에서 수행

- ▶ 자동 완성기능을 사용하여 오버라이드

```
override fun onResume() {  
    super.onResume() //registerListener()메소드로 사용할 센서 등록, 첫 번째 인자는 센서 값을 받을 SensorEventListener이고  
    sensorManager.registerListener( listener: this, 여기서는 this를 지정하여 액티비티에서 센서 값을 받도록 함  
        sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER), //getDefaultSensor() 메서드로 센서 종류를 지정  
        SensorManager.SENSOR_DELAY_NORMAL) //센서 값을 감지하는 속도를 지정  
}
```

- ▶ 센서 값을 너무 자주 읽어오게 되면 리소스를 낭비하고 배터리 소모가 많아 짐

- ▶ SENSOR_DELAY_FASTEST : 가능한한 자주 센서 값을 받음
- ▶ SENSOR_DELAY_GAME : 게임에 적합한 정도로 센서 값을 받음
- ▶ SENSOR_DELAY_NARMAL : 화면 방향이 전환될 때 적합한 정도로 센서 값을 받음
- ▶ SENSOR_DELAY_UI : 사용자 인터페이스를 표시하기에 적합한 정도로 센서 값을 받음

센서를 이용한 수평 측정기

▶ 센서 등록

- ▶ 리스너를 this로 설정했으므로 MainActivity 클래스가 SensorEventListener를 구현하도록 추가

```
class MainActivity : AppCompatActivity(), SensorEventListener {
```

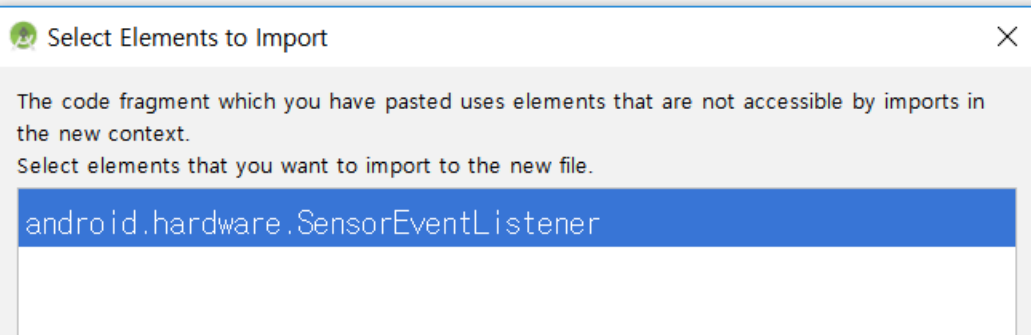
```
    private val
```

```
        getSystem
```

```
    }
```

```
    override fun
```

```
        super.c
```



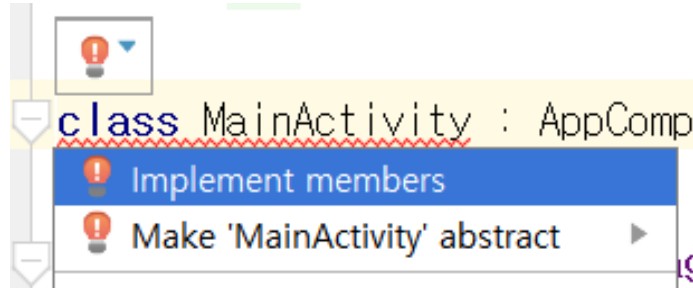
- ▶ 미구현 메서드가 존재하기 때문에 에러 발생

```
class MainActivity : AppCompatActivity(), SensorEventListener {
```


센서를 이용한 수평 측정기

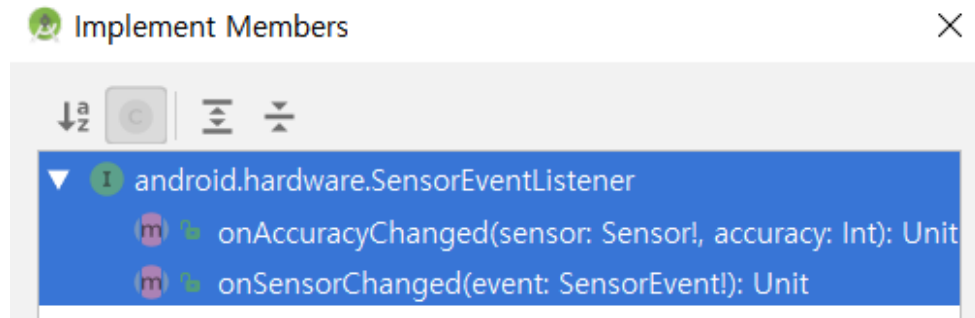
▶ 센서 등록

- ▶ 클래스 이름에 커서를 두면 빨간 전구가 표시됨



- ▶ 구현이 필요한 모든 메서드를 추가

- ▶ Ctrl + A



센서를 이용한 수평 측정기

▶ 센서 등록

▶ 메서드 추가 확인

```
override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int) {  
    TODO( reason: "not implemented") //To change body of created functions use File | Settings | File Templates.  
}
```

```
override fun onSensorChanged(event: SensorEvent?) {  
    TODO( reason: "not implemented") //To change body of created functions use File | Settings | File Templates.  
}
```

▶ AppCompatActivity()

▶ 센서 정밀도가 변경되면 호출

▶ onSensorChanged()

▶ 센서 값이 변경되면 호출되며 SensorEvent 객체에 센서가 측정한 값들과 여러 정보를 읽어옴

센서를 이용한 수평 측정기

▶ 센서 해제

- ▶ 액티비티가 동작 중일 경우에만 센서를 사용하려면 화면이 꺼지기 직전인 onPause() 메서드에서 센서를 해제

```
override fun onPause() {  
    super.onPause()  
    sensorManager.unregisterListener(listener: this)  
}
```

- ▶ unregisterListener() 메서드를 이용하여 센서 사용을 해제할 수 있으며 인자로 SensorEventListener 객체를 지정
 - ▶ MainActivity 클래스에서 해당 객체를 구현할 예정이므로 this로 지정

▶ 센서를 사용할 준비를 마칩

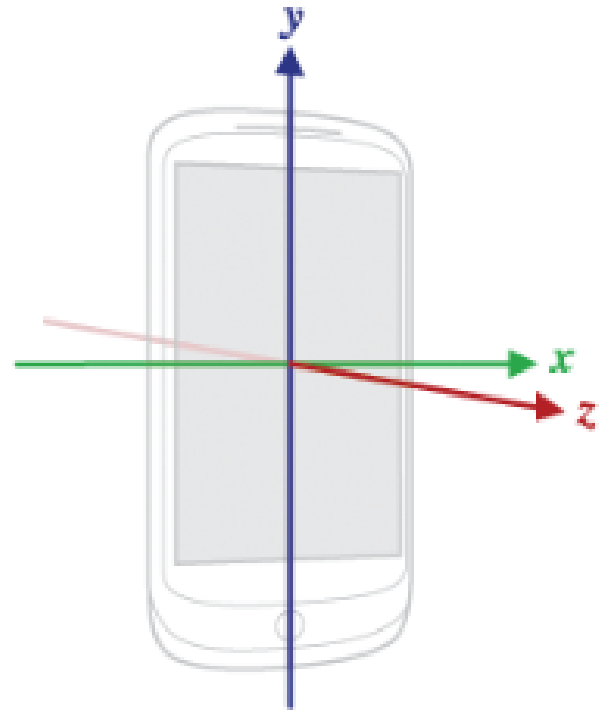
센서를 이용한 수평 측정기

▶ 좌표 시스템

▶ 센서 값을 나타낼 때 x , y , z 표준 3축 좌표계를 사용

▷ 가속도 센서, 중력 센서, 자이로 스코프, 선형 가속도 센서, 자기장 센서

▶ 일반적으로 기기를 정면으로 봤을 때 x 축은 수평이며 오른쪽을 가리키고, y 축은 수직이며 위쪽을, z 축은 화면의 바깥쪽을 향함



센서를 이용한 수평 측정기

▶ 가속도 센서 값 읽어오기

- ▶ 센서는 `SensorEvent` 객체로 값을 넘겨줌
- ▶ `SensorEvent.values[x]` 배열 객체에 센서 값이 담겨 있음
- ▶ 가속도 센서를 사용시 다음과 같이 값을 얻을 수 있음
 - ▷ `SensorEvent.values[0]` : x축 값
 - ▷ `SensorEvent.values[1]` : y축 값
 - ▷ `SensorEvent.values[2]` : z축 값

센서를 이용한 수평 측정기

▶ 가속도 센서 값 읽어오기

▷ onSensorChanged()메서드에 다음과 같이 로그를 표시하도록 코드를 작성 - 오버라이드

- Log.d는 디버그(debug)용 로그를 표시할 경우 사용

- Log.d([태그], [메시지])

```
override fun onSensorChanged(event: SensorEvent?) {  
    event?.let { it: SensorEvent  
        Log.d(tag: "MainActivity", msg: "onSensorChanged: " +  
            "x : ${event.values[0]}, y : ${event.values[1]}, z : ${event.values[2]}")  
    }  
}
```

센서를 이용한 수평 측정기

▶ 가속도 센서 값 읽어오기

▷ Log.d([태그], [메시지])

- 태그 : 로그캣에서 필터링 할 경우 사용
- 메시지 : 출력할 메시지

▷ Log.e() : 에러를 표시

▷ Log.w() : 경고를 표시

▷ Log.i() : 정보성향의 로그를 표시

▷ Log.v() : 모든 로그를 표시

센서를 이용한 수평 측정기

▶ 가로 모드로 고정

- ▶ 수평 측정기는 기기를 가로로 바닥에 놓고 사용
- ▶ 테스트 과정에서 화면의 가로세로 모드가 계속 전환되어 불편함
- ▶ 기기의 방향을 가로로 고정하기 위하여 슈퍼클래스의 생성자를 호출하기 전에 `requestedOrientation` 프로퍼티 값에 가로 방향을 나타내는 `ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE` 값을 설정

```
override fun onCreate(savedInstanceState: Bundle?) {  
  
    requestedOrientation = ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE  
  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_main)  
}
```


센서를 이용한 수평 측정기

▶ 화면이 꺼지지 않게 하기

▶ 절전 기능이 활성화되어 있으면 사용자의 조작이 없을 때 자동으로 화면이 꺼짐

▶ 센서를 테스트하는 동안 화면이 꺼지지 않도록 플래그를 설정

▶ `window.addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON)`

- `Window.addFlags()` 메서드에는 액티비티 상태를 지정하는 여러 플래그를 설정할 수 있음
- `FLAG_KEEP_SCREEN_ON` 플래그를 지정하여 화면이 항상 켜지도록 설정

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

```
    window.addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON)
```

```
    requestedOrientation = ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE
```

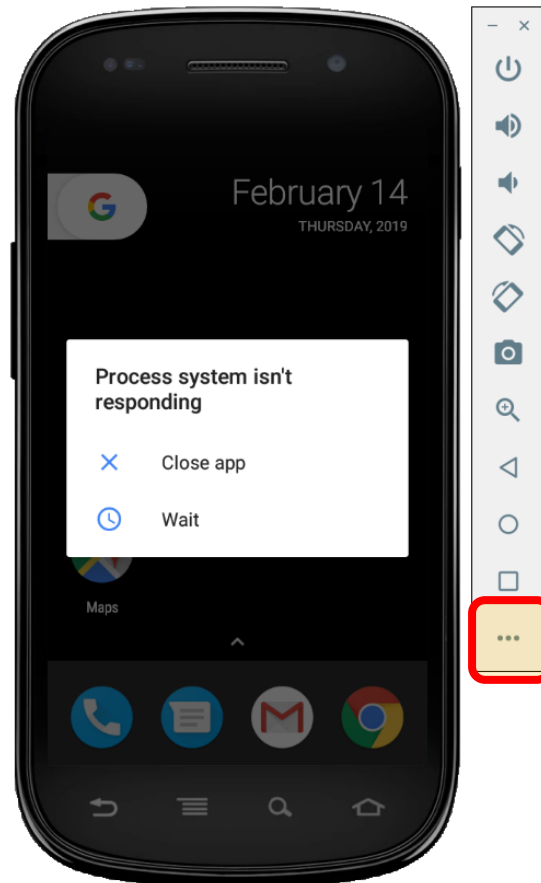
```
    super.onCreate(savedInstanceState)
```

```
    setContentView(R.layout.activity_main)
```

```
}
```

센서를 이용한 수평 측정기

- ▶ 에뮬레이터로 센서 테스트하기
 - ▶ 에뮬레이터로 센서 테스트가 가능
 - ▶ 에뮬레이터 메뉴 중 ... 을 클릭



센서를 이용한 수평 측정기

▶ 커스텀 뷰 작성하기

▶ 사용자가 상황에 따라 임의로 새로운 뷰를 만든 것을 커스텀 뷰(Custom View)라고 함

▶ 예제에서 수평계를 화면에 나타내려면 기존에 없는 뷰를 생성해야 함

▷ 그래픽 API를 사용하여 수평계를 나타내고 센서와 연동되는 뷰를 작성

▶ 구현 순서

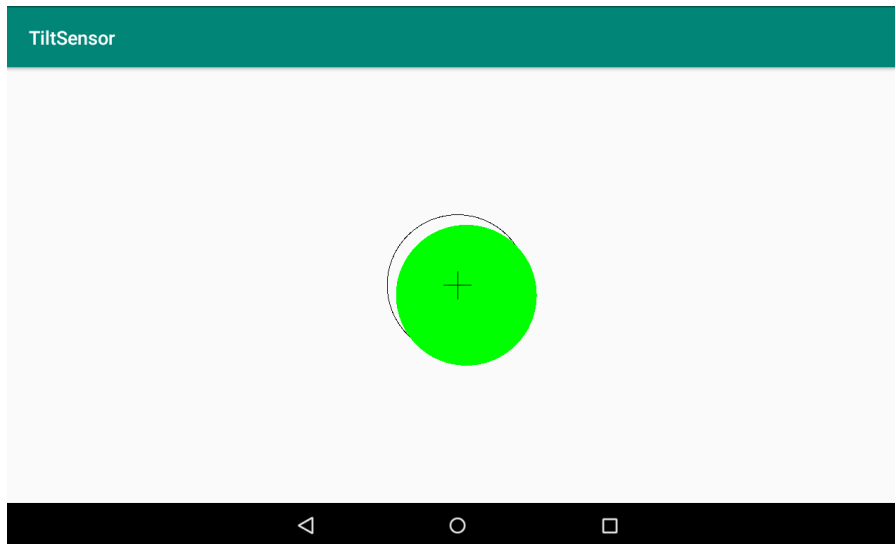
- 1) 수평계 그래픽 표현
- 2) 커스텀 뷰 작성
- 3) 그래픽 API를 사용하여 뷰에 표시
- 4) 센서의 변화 값을 뷰에 반영



센서를 이용한 수평 측정기

▶ 수평계 그래픽 표현

- ▶ 예제에서는 그래픽으로 왼쪽에 있는 둥근 모양 수평계를 표현
- ▶ 다음과 같이 원 2개로 수평계를 표현
- ▶ 기기의 움직임에 따라서 원이 움직임



센서를 이용한 수평 측정기

▶ 커스텀 뷰 작성하기

▶ 기본으로 제공되는 뷰가 아니고 사용자가 새롭게 만드는 뷰

▶ 커스텀 뷰 작성 순서

1) View 클래스를 상속 받은 새로운 클래스를 생성

2) 필요한 메서드를 오버라이드

■ 예제에서는 화면에 그리는 onDraw() 메서드를 오버라이드

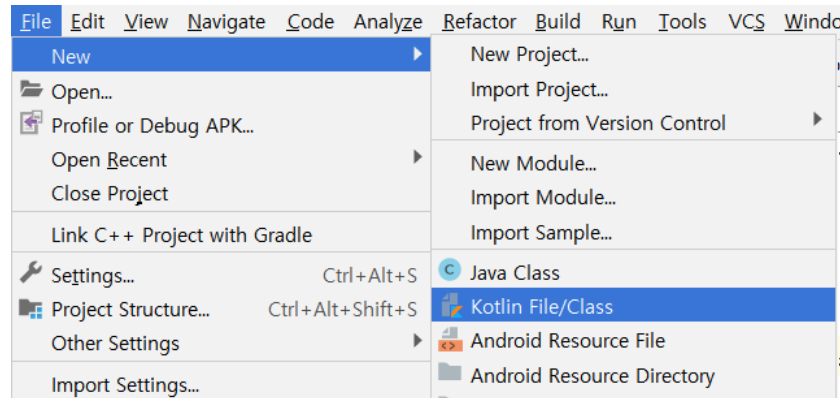
▶ 참고로 이 예제에서는 커스텀 뷰를 생성하고 MainActivity의 onCreate()에서 인스턴스를 생성하여 사용

센서를 이용한 수평 측정기

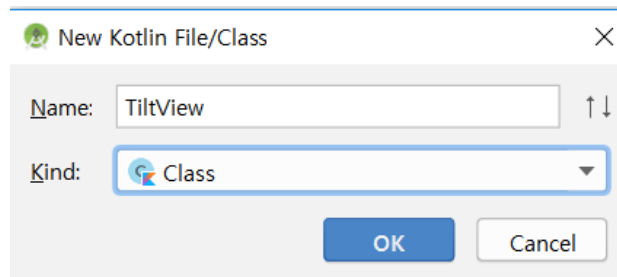
▶ 커스텀 뷰 작성하기

▶ 새로운 클래스를 생성

▷ File - New - Kotlin File/Class



▷ Name : TiltView / Kind : Class



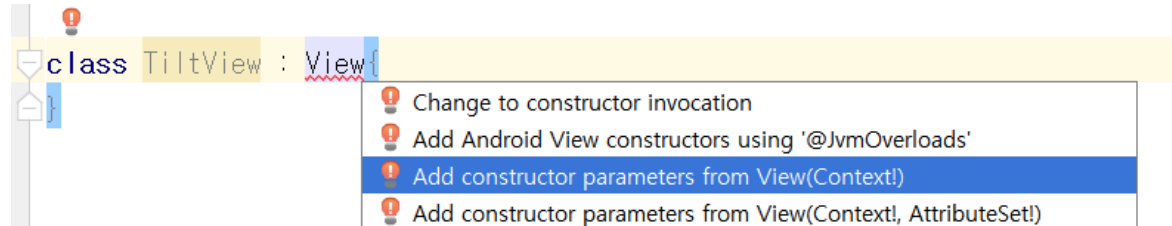
센서를 이용한 수평 측정기

▶ 커스텀 뷰 작성하기

▶ 새로운 클래스는 View를 상속

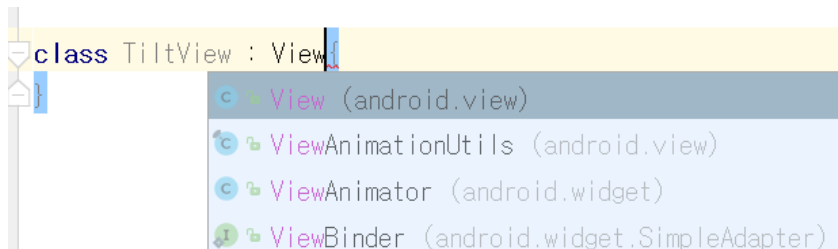
▶ 자동 완성 기능 사용

▶ android.view 패키지의 View를 선택



▶ View 클래스의 생성자 중 최소 한 개를 오버라이드 해야 함

■ Context를 인자로 받는 생성자를 클릭



```
class TiltView(context: Context?) : View(context) {  
}
```

센서를 이용한 수평 측정기

▶ 커스텀 뷰 작성하기

▶ MainActivity.kt 파일에서 TiltView를 생성자를 사용해 인스턴스화하여 화면에 배치

▶ onCreate()메서드 외부에서 선언하고 내부에서 초기화

▷ 선언 : `private lateinit var tiltView: TiltView` //멤버로 늦은 초기화 선언

▷ 초기화 :

```
override fun onCreate(savedInstanceState: Bundle?) {  
  
    window.addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON)  
  
    requestedOrientation = ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE  
  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_main)  
    tiltView = TiltView(context: this) //생성자에 this를 넘겨서 TiltView를 초기화  
    setContentView(tiltView) //activity_main.xml 대신에 tiltView를 화면에 보여줌  
}
```


센서를 이용한 수평 측정기

▶ 그래픽 API

▶ 그래픽을 다루기 위하여 아래와 같은 클래스를 사용


▷ Canvas : 도화지(뷰의 표면)

▷ Paint : 붓(색상, 굵기, 스타일 정의)

▶ 커스텀 뷰에 onDraw()메서드를 오버라이드하여 Canvas 객체를 받아 옴

▷ Canvas 객체에 원하는 그림을 표현

▷ TiltView 클래스 파일에서 자동 완성을 사용하여 onDraw()메서드를 오버라이드

```
override fun onDraw(canvas: Canvas?) {  
      
}
```

센서를 이용한 수평 측정기

▶ 그래픽 API

▶ onDraw() 메서드는 인자로 넘어오는 Canvas 객체에 뷰의 모습을 표현

▶ drawCircle() 메소드 사용 방법

- drawCircle(cx : Float, cy : Float, radius : Float, paint : Paint!) :

///
//!: 널의 허용 여부에 대한 정보를 포함하지 않음

- cx : x의 좌표

- cy : y의 좌표

- Radius : 반지름

- paint : Paint 객체

▶ drawLine(startX: Float, startY : Float, stopX : Float, stopY : Float, paint : Paint!) :

- startX : 한 점의 x 좌표

- startY : 한 점의 y 좌표

- stopX : 다른 점의 x 좌표

- stopY : 다른 점의 y 좌표

- Paint : Paint객체

센서를 이용한 수평 측정기

▶ Paint 객체 초기화

▶ 그리기 메서드에는 Paint 객체가 필요

▶ Paint 객체의 color 및 style을 변경할 수 있음

```
private val greenPaint: Paint = Paint()
```

```
private val blackPaint: Paint = Paint()
```

```
init {  
    // 녹색 페인트  
    greenPaint.color = Color.GREEN //color 프로퍼티는 검정색이 기본값이므로 Color 클래스에 선언된  
                                   색상 중 초록색을 지정  
  
    // 검은색 테두리 페인트  
    blackPaint.style = Paint.Style.STROKE //style 프로퍼티의 속성을 외곽선으로 설정  
}
```

▶ Style 프로퍼티

▶ FILL : 색을 채움, 획 관련 설정 무시(기본 값)

▶ FILL_AND_STROKE : 획과 관련된 설정을 유지하면서 색을 채움

▶ STROKE : 획 관련 설정을 유지하여 외곽선만 그림

센서를 이용한 수평 측정기

▶ 접근 제한자

- ▶ 변수나 함수를 공개할 경우 사용하는 키워드

- ▶ public

 - ▷ 전체 공개 / 기본적으로 public

- ▶ private

 - ▷ 현재 파일(class) 내부에서만 사용 가능

- ▶ internal

 - ▷ 같은 모듈 내에서만 사용가능

 - ▷ 예를 들어 동일한 앱을 스마트폰용, 스마트워치용, TV용으로 만든다면 각 모듈 간에 접근 제한

- ▶ protected

 - ▷ 상속받은 클래스에서만 사용 가능

센서를 이용한 수평 측정기

▶ 수평계 그래픽 표현

▶ 원과 직선으로 수평계를 표현하는 예제

▷ 반지름 : 100 / 십자가 : 길이

▷ 십자가의 한가운데 좌표는 0,0

// 바깥 원

```
canvas?.drawCircle( cx: 0, cy: 0, radius: 100f, blackPaint) // x, y, 반지름, 색상
```

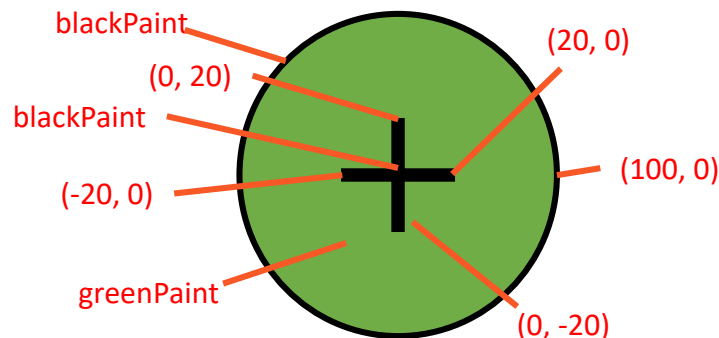
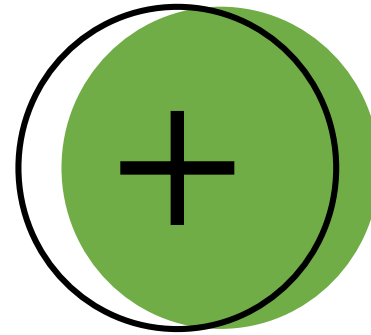
// 녹색 원

```
canvas?.drawCircle( cx: 0, cy: 0, radius: 100f, greenPaint)
```

// 가운데 십자가

```
canvas?.drawLine( startX: -20, startY: 0, stopX: 20, stopY: 0, blackPaint) // x1, y1, x2, y2, 색상
```

```
canvas?.drawLine( startX: 0, startY: -20, stopX: 0, stopY: 20, blackPaint) // 가로, 세로, 색상
```

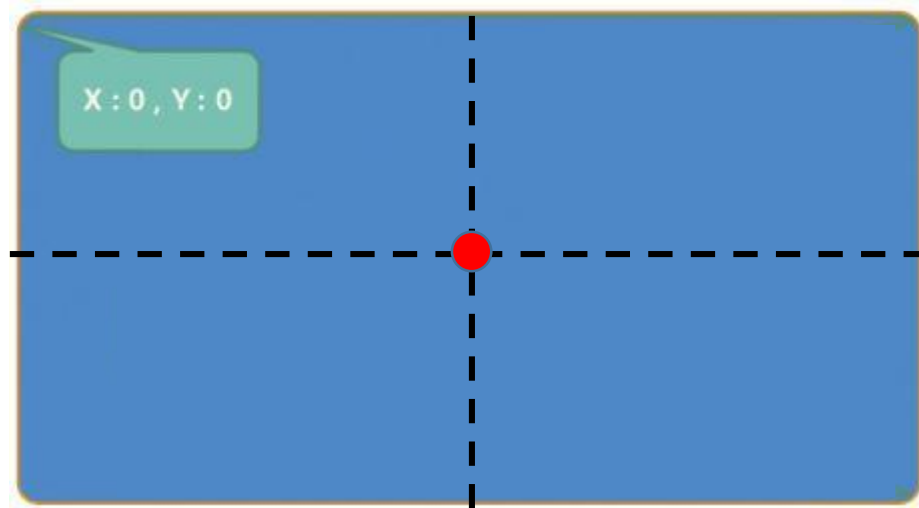


센서를 이용한 수평 측정기

▶ 수평계 그래픽 표현

▶ 원과 직선으로 수평계를 표현하는 예제

- ▶ 앞의 예제 코드 응용하여 onDraw() 메소드에 좌표와 크기를 입력
- ▶ 안드로이드 좌표계는 왼쪽 상단이 (0, 0)
- ▶ 중점이 (cX, cY)라면 $(width / 2, height / 2)$ 라는 공식이 성립



센서를 이용한 수평 측정기

▶ 수평계 그래픽 표현

▶ View의 크기 정보를 가져오도록 `onSizeChanged()` 메서드를 오버라이드

▷ 뷰의 크기가 변경될 때마다 호출됨

▶ 중점의 좌표를 구할 수 있도록 `TiltView` 클래스에 아래 코드 추가

```
private var cX: Float = 0f // 영에프
```

```
private var cY: Float = 0f
```

```
override fun onSizeChanged(w: Int, h: Int, oldw: Int, oldh: Int) {  
    cX = w / 2f  
    cY = h / 2f  
}
```

▷ `w` : 변경된 가로 길이

▷ `h` : 변경된 세로 길이

▷ `oldw` : 변경 전 가로 길이

▷ `oldh` : 변경 전 세로 길이

센서를 이용한 수평 측정기

▶ 수평계 그래픽 표현

▶ onDraw에 아래 코드 추가

▶ 중점 값을 받아와서 원과 십자가를 표시

```
override fun onDraw(canvas: Canvas?) {  
    // 바깥 원  
    canvas?.drawCircle(cX, cY, radius: 100f, blackPaint)  
    // 녹색 원  
    canvas?.drawCircle(cX, cY, radius: 100f, greenPaint)  
    // 가운데 십자가  
    canvas?.drawLine(startX: cX - 20, cY, stopX: cX + 20, cY, blackPaint)  
    canvas?.drawLine(cX, startY: cY - 20, cX, stopY: cY + 20, blackPaint)  
}
```


센서를 이용한 수평 측정기

▶ 센서 값을 뷰에 반영

▶ 안드로이드 기기가 움직일 때 감지되는 센서 값에 따라서 녹색원이 움직이도록 구현

▶ SensorEvent를 인자로 받는 onSensorEvent()메서드를 정의

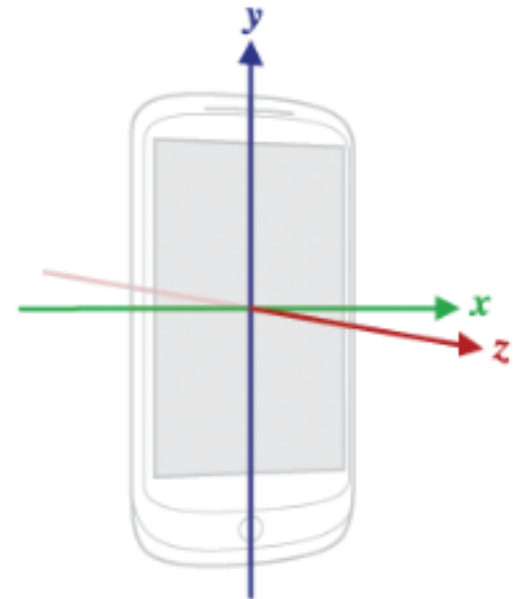
```
fun onSensorEvent(event: SensorEvent) { // SensorEvent 값을 인자로 받음
    // 화면을 가로로 돌렸으므로 X축과 Y축을 서로 바꿈
    yCoord = event.values[0] * 20 // x, y값을 바꿔야 확인하기 편리
    xCoord = event.values[1] * 20 // 이벤트 값이 작기때문에 원의 움직임을 증가시키려 20을 곱함

    invalidate() // onDraw() 메서드를 다시 호출하는 메서드
                  // 즉, 뷰를 다시 그림
}
```

▶ yCoord와 xCoord는 멤버로 선언

```
private var xCoord: Float = 0f
```

```
private var yCoord: Float = 0f
```



센서를 이용한 수평 측정기

▶ 센서 값을 뷰에 반영

- ▶ 녹색원의 위치를 반영하기 위하여 x좌표와 y좌표에 센서 값을 더함

```
override fun onDraw(canvas: Canvas?) {  
    // 바깥 원  
    canvas?.drawCircle(cX, cY, radius: 100f, blackPaint)  
    // 녹색 원  
    canvas?.drawCircle( cx: cX + xCoord, cy: cY + yCoord, radius: 100f, greenPaint)  
    // 가운데 십자가  
    canvas?.drawLine( startX: cX - 20, cY, stopX: cX + 20, cY, blackPaint)  
    canvas?.drawLine(cX, startY: cY - 20, cX, stopY: cY + 20, blackPaint)  
}
```

센서를 이용한 수평 측정기

▶ 센서 값을 뷰에 반영

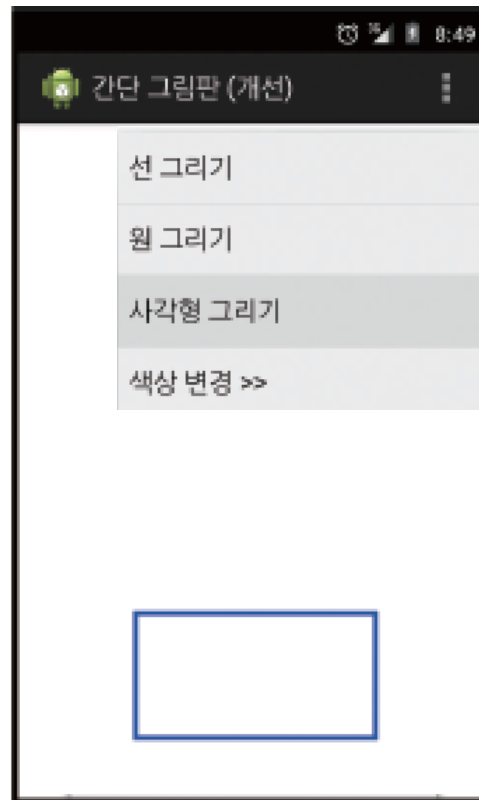
- ▶ 액티비티에서 센서 값이 변경되면 `onSensorChanged()` 메서드가 호출되므로 내부에서 `TiltView`에 센서 값을 전달

```
override fun onSensorChanged(event: SensorEvent?) {  
    event?.let { it: SensorEvent  
        Log.d( tag: "MainActivity", msg: "onSensorChanged: " +  
            "x : ${event.values[0]}, y : ${event.values[1]}, z : ${event.values[2]}")  
  
        tiltView.onSensorEvent(event)  
    }  
}
```

- ▶ 앱을 실행하여 수평계가 잘 동작하는지 확인

응용 + 복습 + 실습문제

- ▶ 다음 그림과 같이 코드를 작성하시오.
 - ▶ 옵션 메뉴에서 선택한 특정 그림이 나타나도록 하시오.
 - ▶ 색상이 옵션 메뉴에서 선택되도록 하시오.
 - ▶ 색상은 서브 메뉴로 나오게 하고 빨강 초록 파랑 세가지만 사용



Q & A
