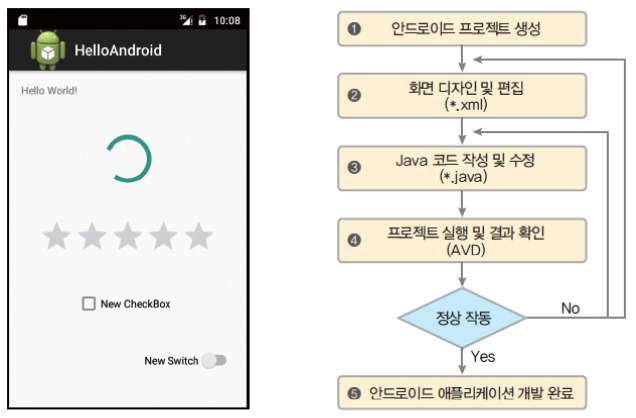
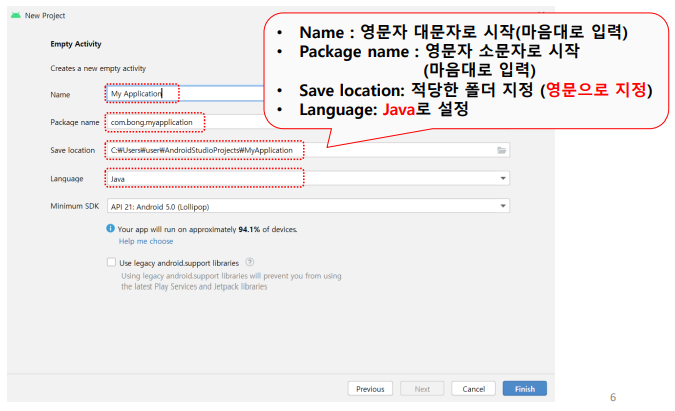
**7주차 안드로이드 (프로젝트 만들기)**

**안드로이드 프로젝트 개발 단계**



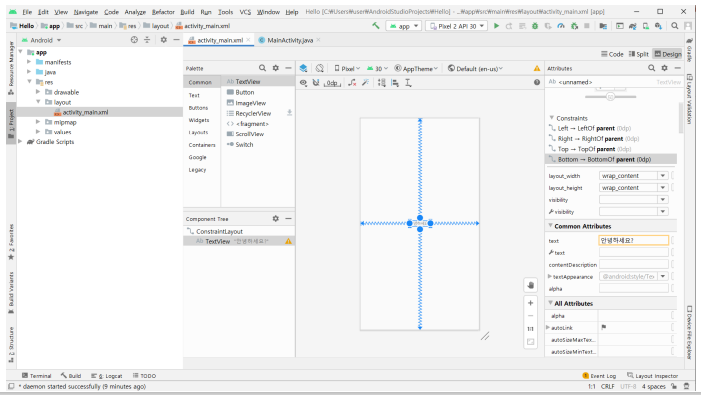
**프로젝트 생성**

* [Configure Your Project] 창에서 프로젝트 정보 입력

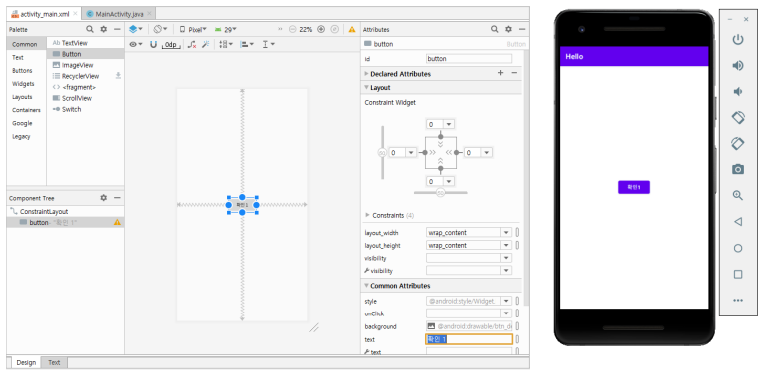


**Hello 프로젝트 하나씩 바꾸어 보기**

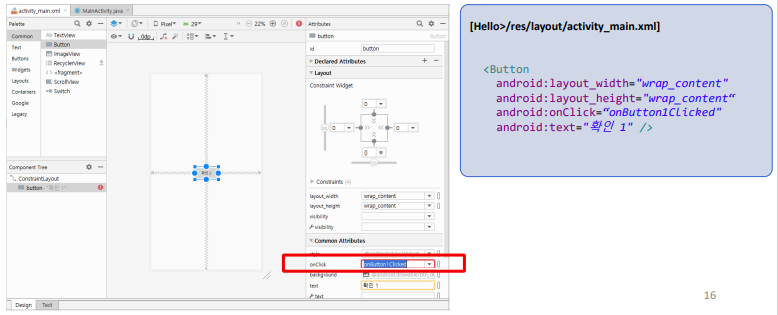
* 화면 중앙에 있는 메시지 변경하기
  + TextView 태그의 text 속성을 바꾸면 글자를 바꿀 수 있음



* 화면에 버튼 추가하기



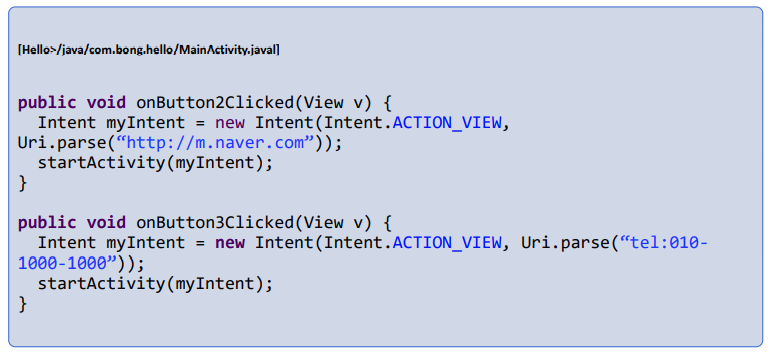
* 버튼 눌렀을 때 메시지가 나타나게 하기
  1. XML 레이아웃 파일의 버튼에 onClick 속성 값 넣기
  2. 소스파일에 이벤트 처리 함수 추가하기





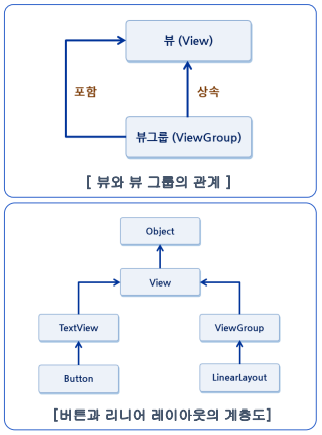
**더 많은 활용**

* 자바 소스파일에 메소드 추가(새로운 버튼 2개)



**9주차 안드로이드 살펴보기**

**뷰와 뷰그룹의 정의**



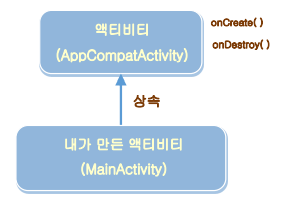
* 뷰(View)
  + 화면에 보이는 모든 것
  + UI 구성 요소
* 뷰 그룹(View Group)
  + 뷰들을 여러 개 포함하고 있는 것
  + 뷰 그룹도 뷰에서 상속하여 뷰가 됨
* 위젯(Widget)
  + 뷰 중에서 일반적인 컨트롤의 역할을 하고 있는 것
  + 버튼, 텍스트 등등
* 레이아웃(Layout)
  + 뷰 그룹 중에서 내부에 뷰들을 포함하고 있으면서 그것들을 배치하는 역할을 하는 것

**XML 레이아웃의 구성**

* 뷰 태그와 속성으로 구성됨



**상속**

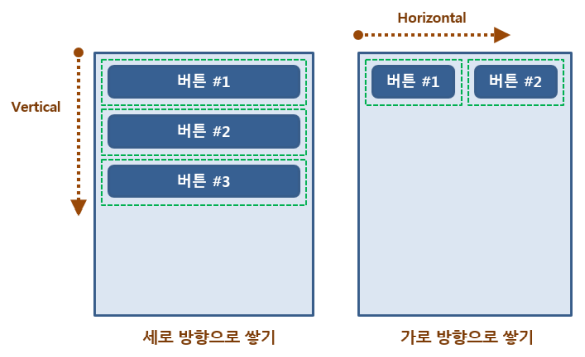


* 상속
  + 객체지향의 가장 기본적인 개념 중 하나
  + 부모의 특성을 그대로 물려받는 것으로 변수나 메소드 재사용 가능
* 액티비티의 상속
  + extends 키워드 사용  
    public class MainActivity extends AppCompatActivity
* 부모 클래스의 메소드를 재정의
  + onCreate() 메소드는 이미 부모 클래스에 정의되어 있음
  + 기능을 추가하고 싶을 때 재정의(Override)
* this와 super
  + 나 자신은 this, 부모는 super를 사용하여 변수나 메소드 참조
  + super.onCreate( … );

**레이아웃**

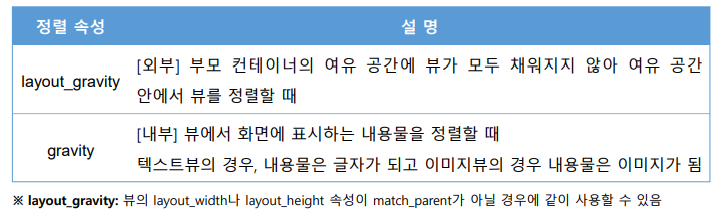
* 제약 레이아웃(ConstraintLayout)
  + 제약 조건(Constraint) 기반 모델
  + 제약 조건을 사용해 화면을 구성하는 방법
  + 안드로이드 스튜디오에서 자동으로 설정하는 디폴트 레이아웃
* 리니어 레이아웃(LinearLayout)
  + 박스(Box) 모델
  + 한 쪽 방향으로 차례대로 뷰를 추가하며 화면을 구성하는 방법
  + 뷰가 차지할 수 있는 사각형 영역을 할당
* 상대 레이아웃(RelativeLayout)
  + 규칙(Rule) 기반 모델
  + 부모 컨테이너나 다른 뷰와의 상대적 위치로 화면을 구성하는 방법
* 프레임 레이아웃(FrameLayout)
  + 싱글(Single) 모델
  + 가장 상위에 있는 하나의 뷰 또는 뷰그룹만 보여주는 방법
  + 여러 개의 뷰가 들어가면 중첩하여 쌓게 됨. 가장 단순하지만 여러 개의 뷰를 중첩한 후 각 뷰를 전환하여 보여주는 방식으로 자주 사용함
* 테이블 레이아웃(TableLayout)
  + 격자(Grid) 모델
  + 격자 모양의 배열을 사용하여 화면을 구성하는 방법
  + HTML에서 많이 사용하는 정렬 방식과 유사하지만 많이 사용하지는 않음

**리니어 레이아웃 사용방식**

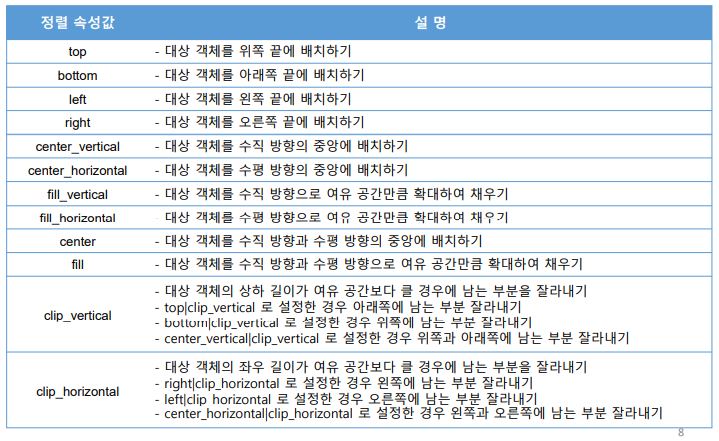


**리니어 레이아웃 – 뷰 정렬하기**

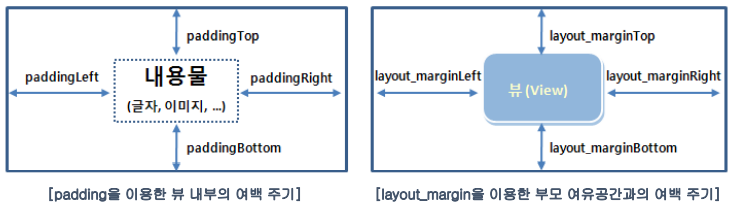
* 두 가지 정렬 속성



**리니어 레이아웃 – 정렬을 위해 사용할 수 있는 값**

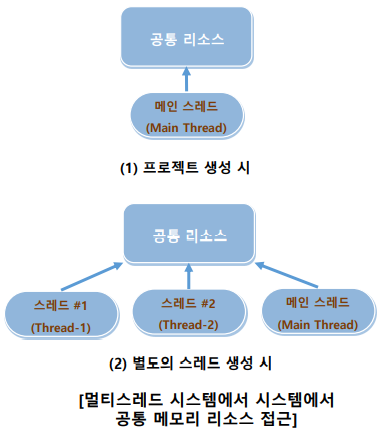


**리니어 레이아웃 – 마진과 패딩 설정**



* padding 속성
  + 뷰 안의 내용물인 텍스트나 이미지와 뷰 안의 영역 사이의 여백을 줄 수 있는 방법
* layout\_margin 속성
  + 부모 컨테이너와 여유 공간과 뷰 사이의 여백을 줄 수 있는 방법

**멀티 스레드**

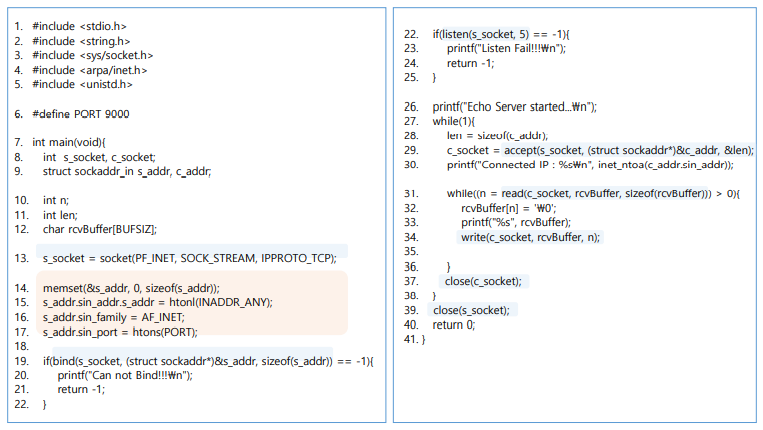


* 메인 액티비티
  + 애플리케이션이 실행될 때 하나의 프로세스에서 처리
  + 이벤트를 처리하거나 필요한 메소드를 정의하여 기능을 구현하는 경우에도 동일한 프로세스 내에서 실행
* 문제점
  + 대기 시간이 길어지는 네트워크 요청 등의 기능을 수행할 때는 화면에 보이는 UI도 멈춤 상태로 있게 됨
* 해결 방안
  + 하나의 프로세스 안에서 여러 개의 작업이 동시에 수행하는 멀티 스레드 방식을 사용
* 멀티 스레드
  + 같은 프로세스 안에 들어 있으면서 메모리 리소스를 공유하게 되므로 효율적인 처리가 가능
  + 동시에 리소스를 접근할 경우 문제 발생
  + 안드로이드에서는 main스레드에서만 UI 접근 가능

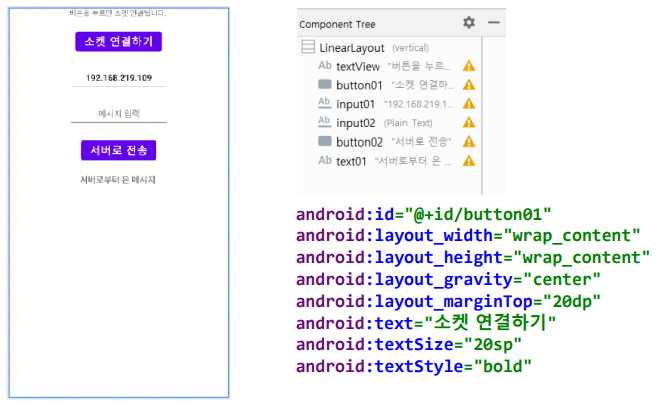
**핸들러 사용하기**

* 스레드 사용
  + 스레드는 동일 프로세스 내에 있기 때문에 작업 수행의 결과를 바로 처리할 수 있음
  + 그러나 UI 객체는 직접 접근할 수 없으므로 핸들러(Handler) 객체를 사용함

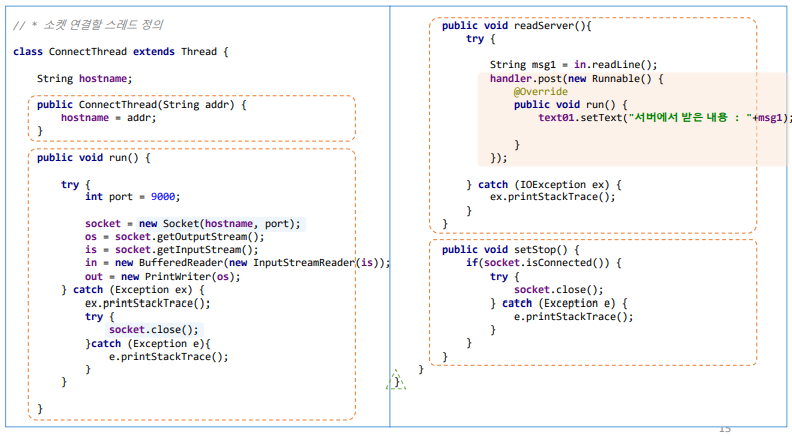
**안드로이드 소켓 통신 예제\_서버**



**안드로이드 소켓 통신 예제\_클라이언트**







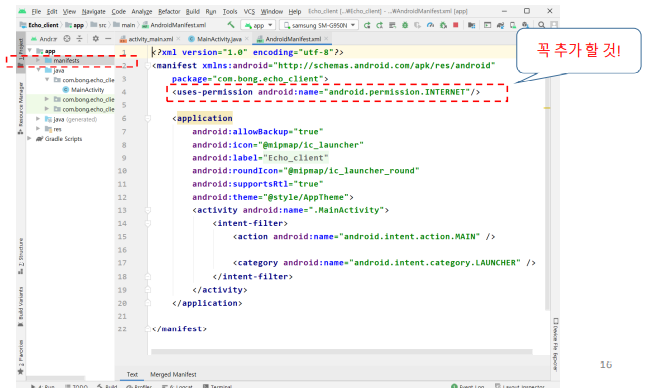
**XML**





**AndroidManifast**

* 설치된 앱의 구성요소가 어떤 것인지, 어떤 권한이 부여되었는지 시스템에게 알려줌

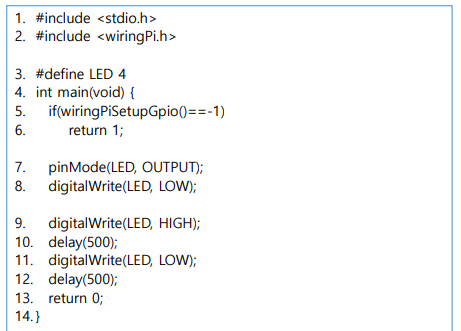


**네트워킹 사용 시 주의할 점**

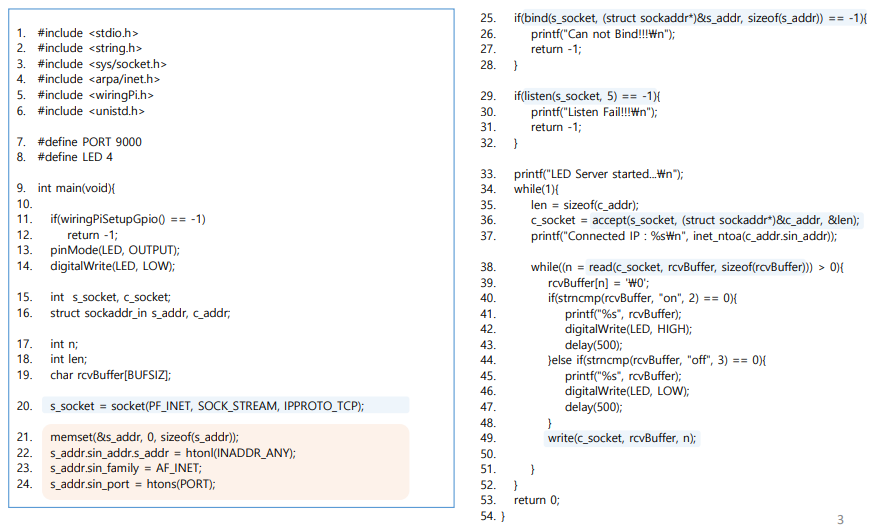
* 네트워킹을 사용할 때에는 반드시 스레드 사용
  + 최신 버전의 안드로이드에서는 네트워킹을 사용할 때에는 반드시 스레드를 사용하도록 변경되었음 (이전에는 스레드 없이도 가능했음)
* UI 업데이트를 위해서는 반드시 핸들러 사용
  + 네트워킹을 위해 새로 만든 스레드 안에서 그 결과를 보여주기 위해 UI 업데이트를 하는 경우 스레드 부분에서 공부한 바와 같이 핸들러를 사용해야 함

**10주차 안드로이드로 LED 켜고 끄기**

**라즈베리파이\_LED 제어**



**LED 소켓 통신\_서버 (led\_server.c)**



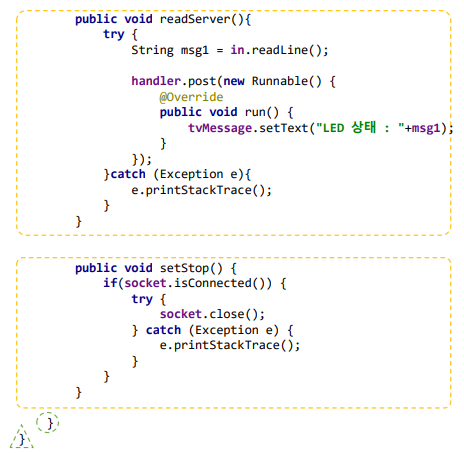
**LED 소켓 통신\_클라이언트 (led\_client)**



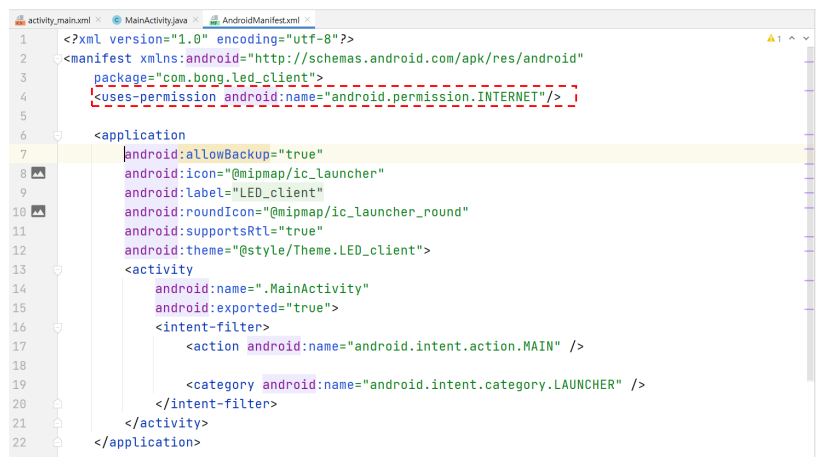
**안드로이드 소켓 통신 예제\_클라이언트**







**AndroidManifest**



**XML**



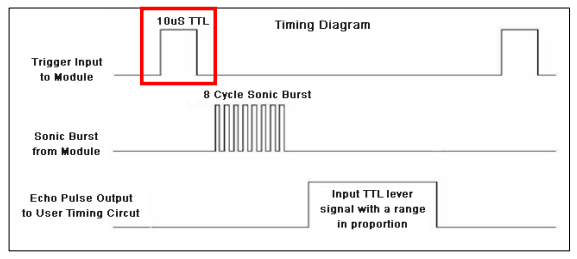


**11장 안드로이드\_초음파센서**

**초음파 센서 동작 방식**

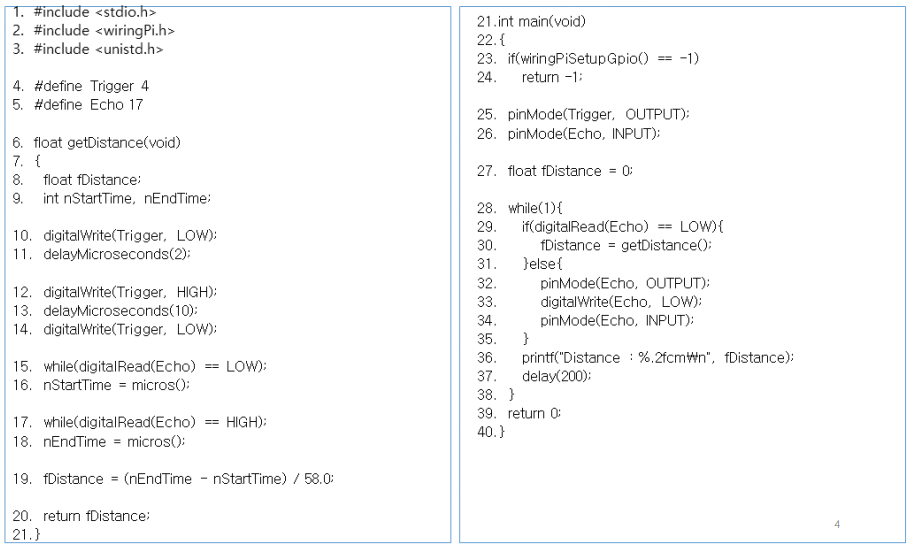
* 초음파를 발생하고 물체에 부딪혀 돌아오는 시간을 측정함으로써 거리를 센싱
  + Trig 핀 출력 신호와 Echo 핀 입력 신호를 이용
  + 최소 10us HIGH 레벨 신호를 통한 IO trigger(Trig핀이 출력)를 사용하며, 모듈은 자동으로 8번의 40kH 신호를 보내고 펄스 신호가 돌아오는지 여부를 검출
  + 만약, HIGH 레벨을 통해 신호가 돌아오면(Echo핀이 감지), HIGH 출력 IO 지속 시간은 되돌아 오는 초음파가 보내지는 시간
  + 시험 거리는 다음과 같이 계산
    - 초음파 속도: 340 m/s
    - us / 58 = centimeters

**타이밍 다이어그램**



* 센싱을 시작할 때, 트리거 입력에 짧은 10uS 펄스를 제공
* 모듈은 40kHz의 초음파의 버스트(Burst)를 8차례 전송하고, 그것의 에코를 수신
* 전송하는 트리거 신호와 받는 에코 신호 사이의 시간 간격을 통하여 그 범위를 계산
* 센싱이 시작되고 다음 센싱이 시작되기까지의 간격, 즉, 측정 주기는 60ms 이상으로 사용하며, 이는 에코 신호 중에 트리거 신호가 전달되는 것을 방지하기 위함

**라즈베리파이\_초음파 센서**

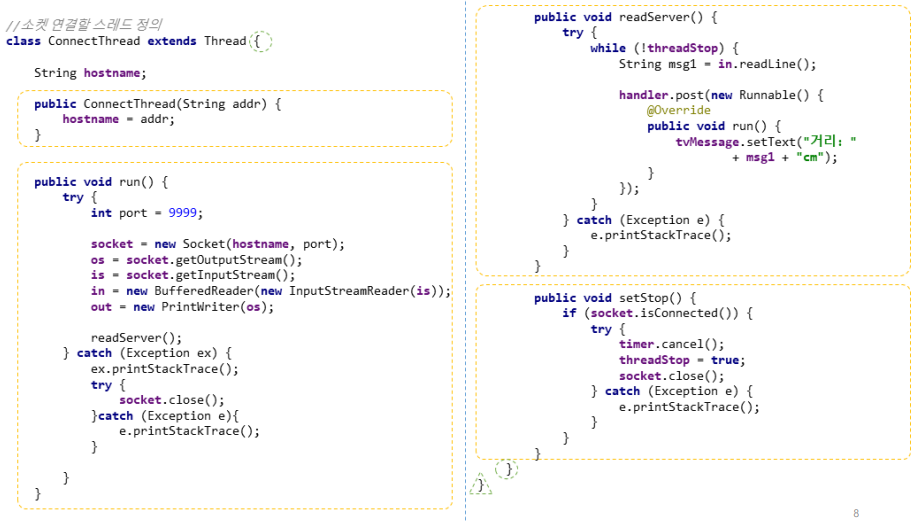


**초음파 센서\_클라이언트 (ultrasonic\_client)**

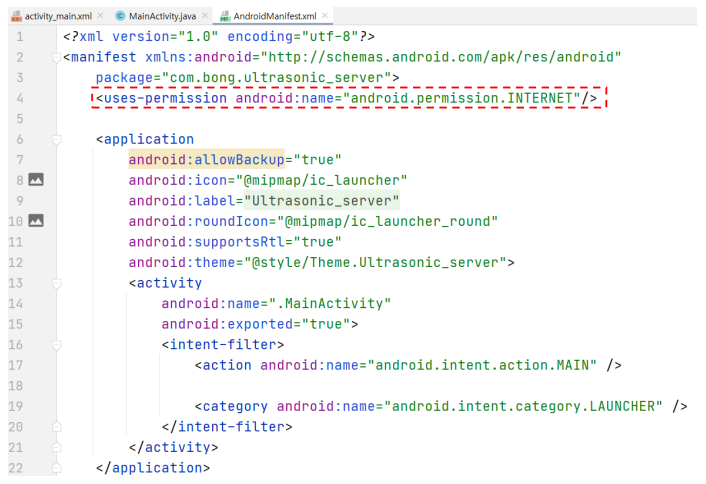








**AndroidManifest**



**XML**





**12주차 안드로이드\_온습도 센서**

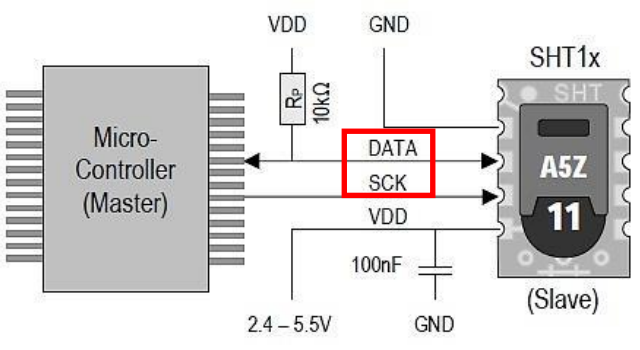
**온/습도 센서**

온도와 습도 등을 측정하는 센서

* + 온/습도 센서에서 받아들인 정보는 하드웨어를 통하여 디지털화된 결과를 도출
* SHT11(온/습도) 센서
  + SENSIRION사의 SHT1x 시리즈의 SHT11 Humidity & Temperature Sensor를 사용
  + 디지털 출력 온/습도 센서인 SHT11은 하나의 칩에 두 개의 센서를 집적화 한 제품으로 특징은 다음과 같음
    - 측정 범위: 0~100% RH
    - 상대 습도 정밀도 +/- 3%
    - 온도 정밀도: +/- 0.4도 @ 25도
    - 보정된 디지털 출력 (2 와이어 인터페이스)
    - 빠른 반응 속도 < 4sec.
    - 저전력 소비
    - 낮은 가격
    - 대량 생산용으로 디자인 됨, 가격이 민감한 응용 제품용
    - 뛰어난 장기 안정성
    - 보정과 디지털 2 와이어 인터페이스 기능으로 사용이 쉬움

**온/습도 센서(내부의 A/D 변환기)와 라즈베리 파이의 통신**

* 센싱된 온습도의 아날로그 수치는 14비트 A/D 변환기를 통해 디지털로 수치화하고 I2C를 통하여 정보를 처리할 MCU(라즈베리파이)로 전달
* 일반적인 회로구성 예

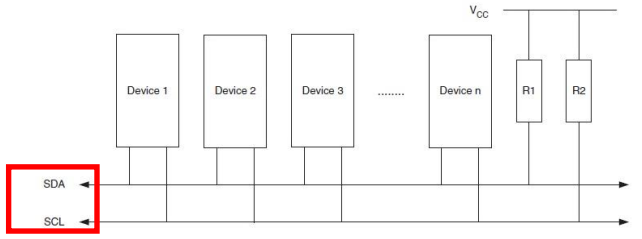


**온/습도 센서 동작 방식**

* 센서가 전원을 입력 받으면 온/습도를 센싱하는 하드웨어를 통해 아날로그 정보를 측정
* 측정된 정보는 센서 자체가 가진 A/D 변환 기능을 통해 디지털 수치로 변환
* 변환된 수치는 직렬 버스를 통해 CPU로 전달

**2-Wire 직렬 인터페이스(Two-wire Serial Interface, TWI)**

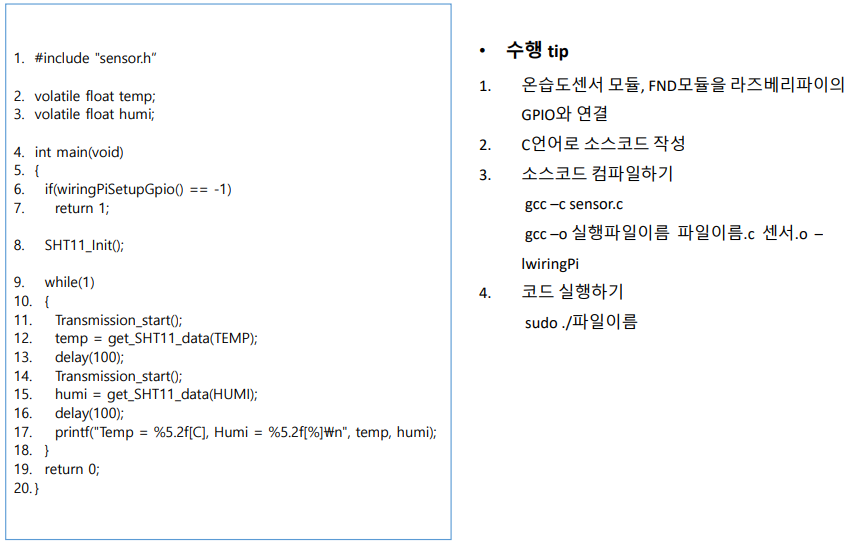
* TWI 프로토콜은 단지 2개의 양방향 버스 라인, 클럭(SCL)을 위한 하나의 버스와 데이터(SDA)를 위한 또 하나의 버스를 이용한 128개 이상 다른 장치의 상호연결을 위한 시스템 디자이너를 허용
  + 디바이스를 구분하기 위해 주소 개념 사용



**TWI(I2C) 특징**

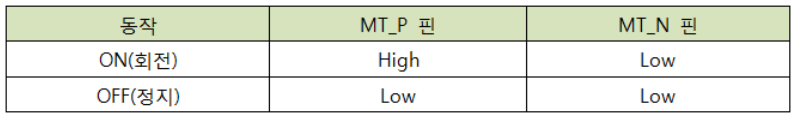
* 단순하지만 파워풀하고 유연한 통신 인터페이스, 단지 2개 버스 라인 필요
* 마스터와 슬레이브 동작을 지원
* 장치는 전송자와 수신자로써 동작할 수 있음
* 클럭 신호를 단방향이며, 마스터에 의해 발생되고 데이터 신호는 양방향으로 데이터를 송수신하기 위해 사용

**라즈베리파이\_동작 센서(temp.c) sensor.h / sensor.c**

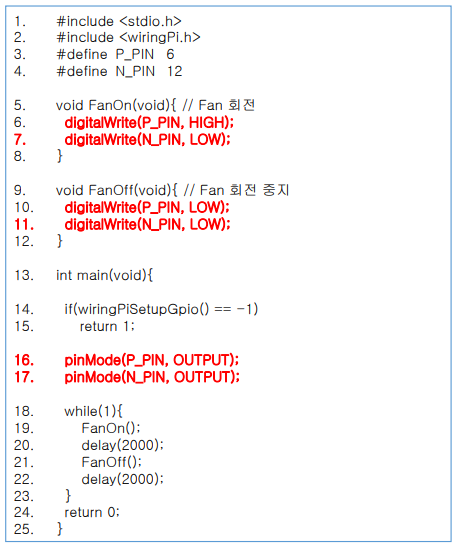


**Fan 개념 및 제어방법**

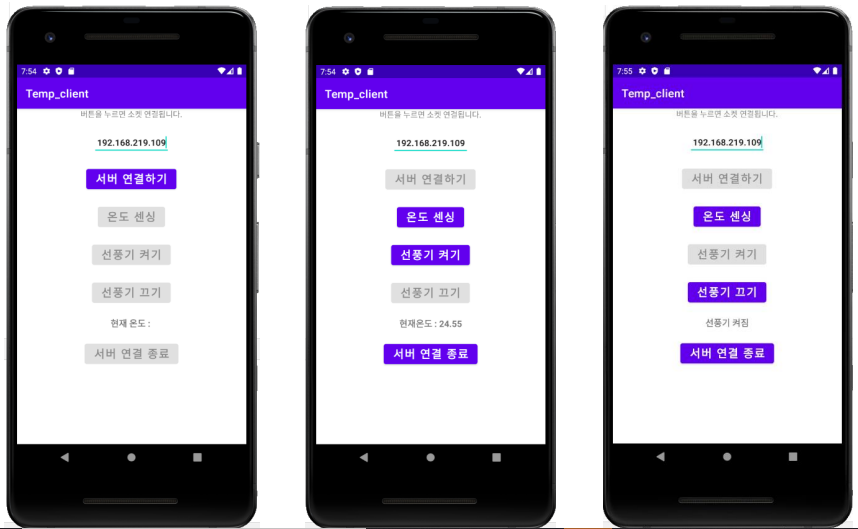
* 전력(전기적 에너지)을 회전운동의 힘(기계적 에너지)으로 바꾸어 바람, 즉 공기의 흐름을 만드는 기계로써, 전력을 공급하면 회전 날개를 회전
  + 선풍기에 FAN이 포함되고, 컴퓨터도 칩의 열을 식혀주기 위해 작은 FAN이 들어있으며, 내부의 공기를 환기하는 목적의 FAN도 있음
* FAN의 동작은 ON/OFF 두 가지
* Fan 모듈의 FAN 포트 각 핀에 입력되는 신호에 따른 동작



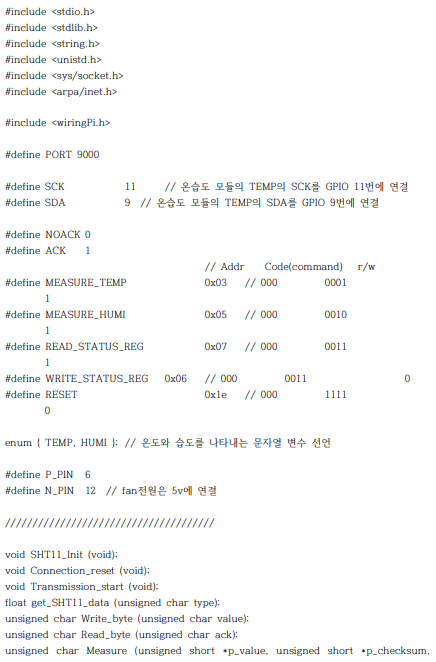
**라즈베리파이\_FAN**

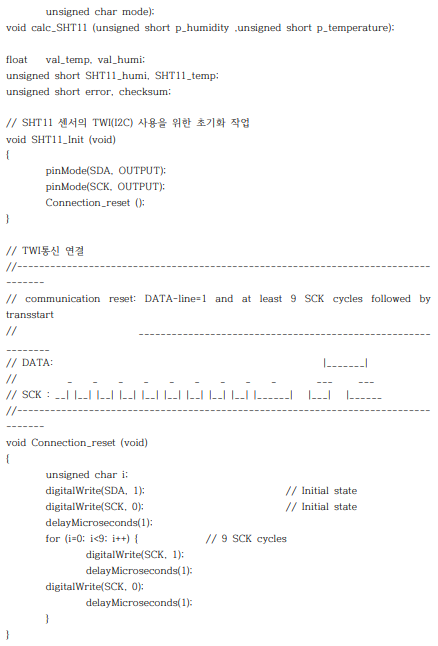


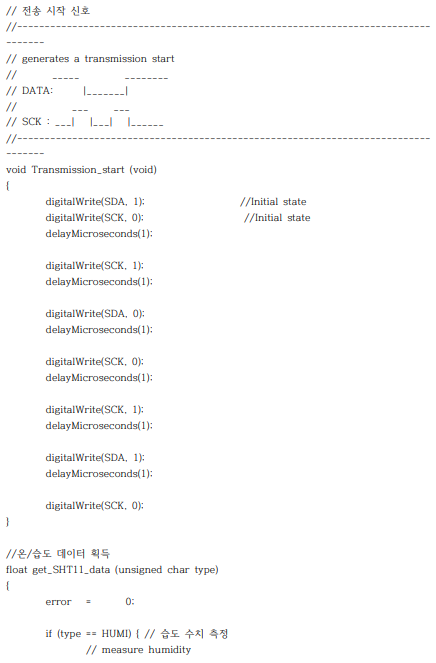
**동작 센서\_클라이언트 (Temp\_client)**

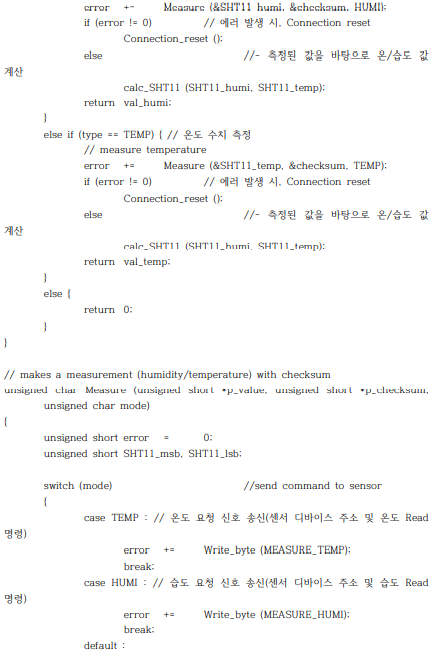


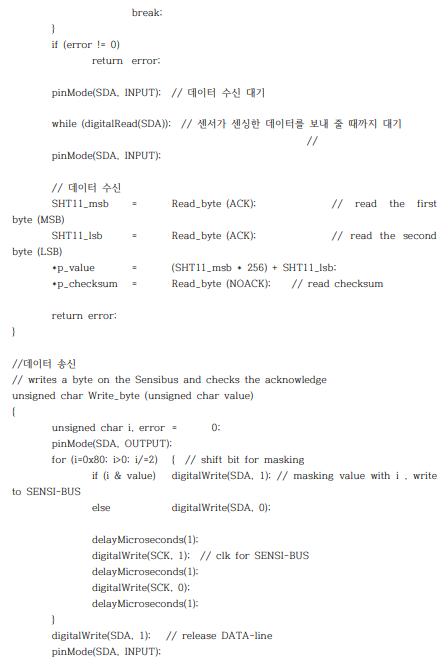
**온/습도 서버**

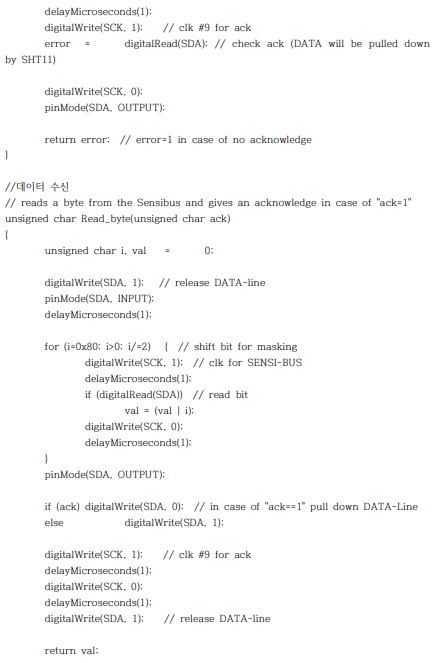


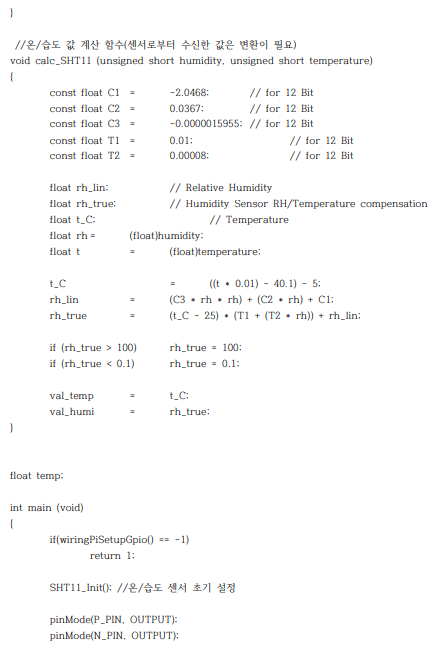


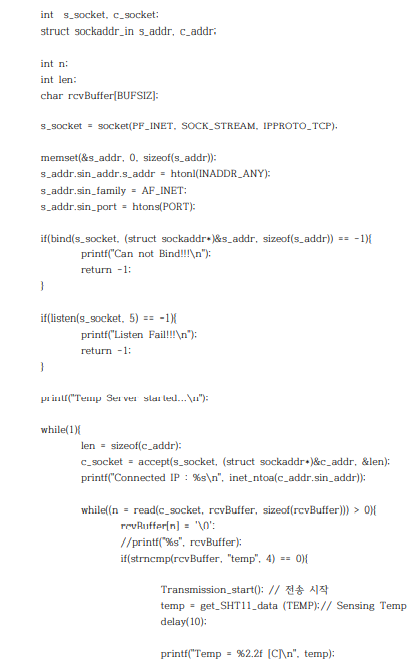


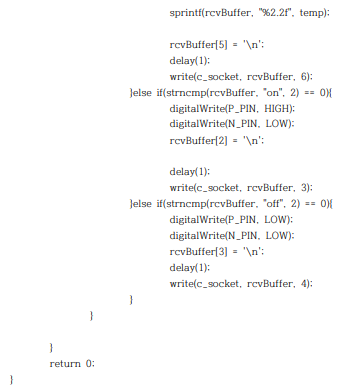












**온/습도 클라이언트**





