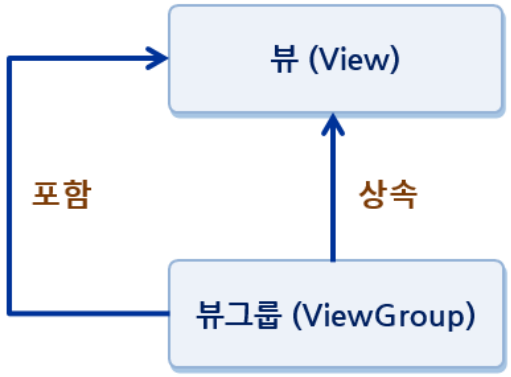
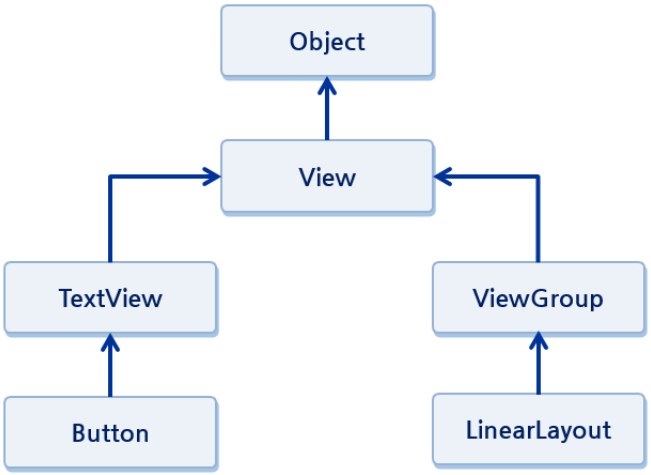
안드로이드 살펴보기

뷰와 뷰그룹의 정의



[뷰와 뷰 그룹의 관계]



[버튼과 리니어 레이아웃의 계층도]

- 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

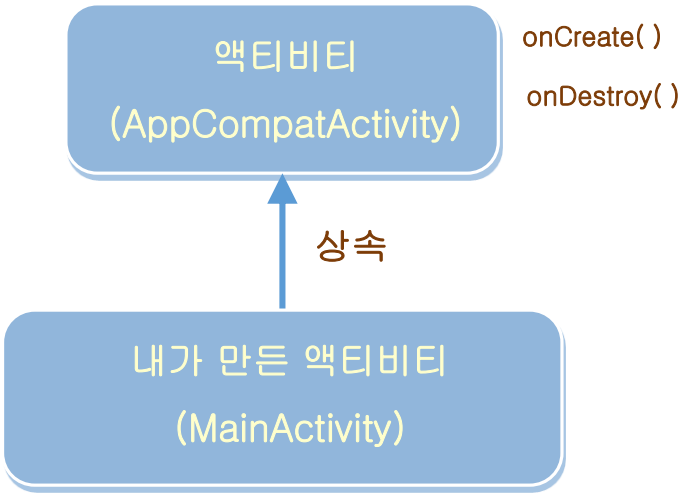
- 뷰 태그와 속성으로 구성됨

```
<시작 태그>
  속성1="속성값1"
  속성2="속성값2"
  ...
</끝 태그>
```

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!" />
```

```
<android.support.constraint.ConstraintLayout>
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    …
</android.support.constraint.ConstraintLayout>
```

상속



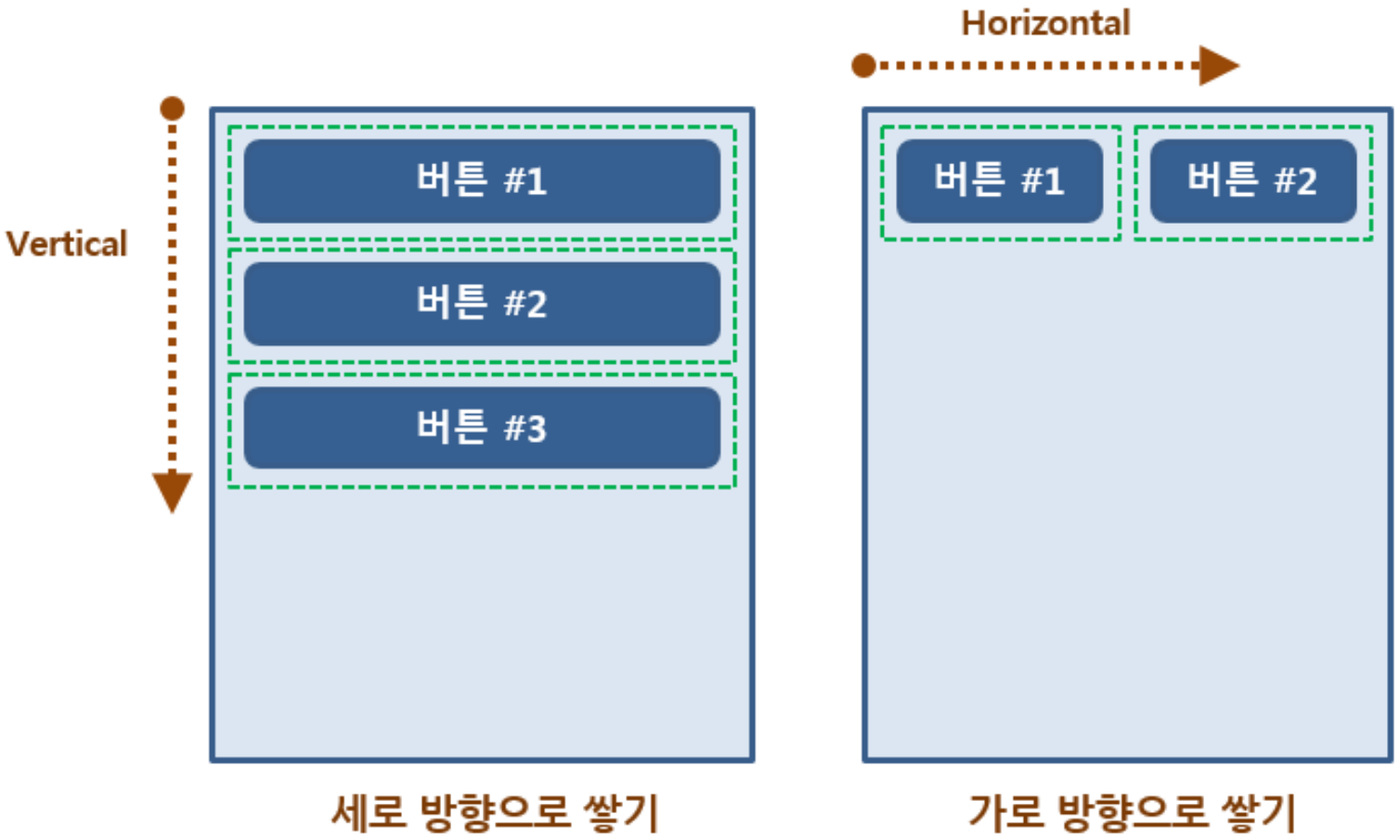
- 상속
  - 객체지향의 가장 기본적인 개념 중 하나
  - 부모의 특성을 그대로 물려받는 것으로 변수나 메소드 재사용 가능
- 액티비티의 상속
  - extends 키워드 사용

    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 super 를 사용하여 변수나 메소드 참조
  - super.onCreate(…);

레이아웃

레이아웃 이름	설 명
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

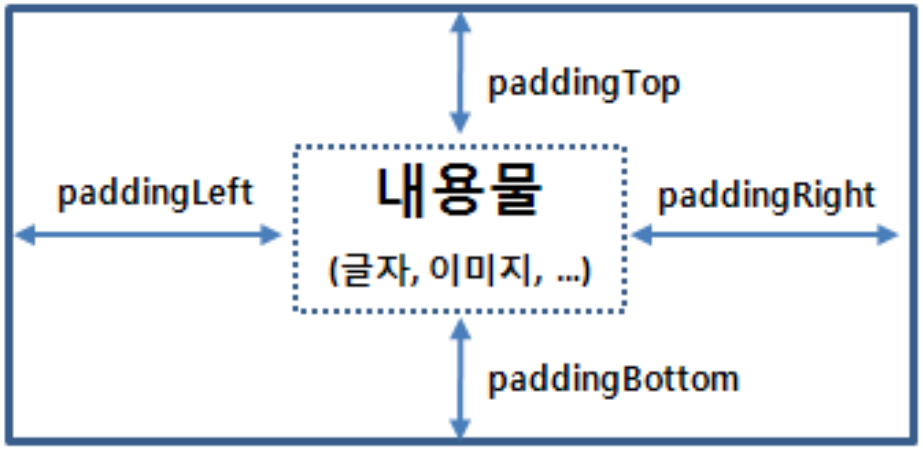
정렬 속성	설 명
layout_gravity	[외부] 부모 컨테이너의 여유 공간에 뷰가 모두 채워지지 않아 여유 공간 안에서 뷰를 정렬할 때
gravity	[내부] 뷰에서 화면에 표시하는 내용물을 정렬할 때 텍스트뷰의 경우, 내용물은 글자가 되고 이미지뷰의 경우 내용물은 이미지가 됨

※ **layout_gravity:** 뷰의 layout_width나 layout_height 속성이 match_parent가 아닐 경우에 같이 사용할 수 있음

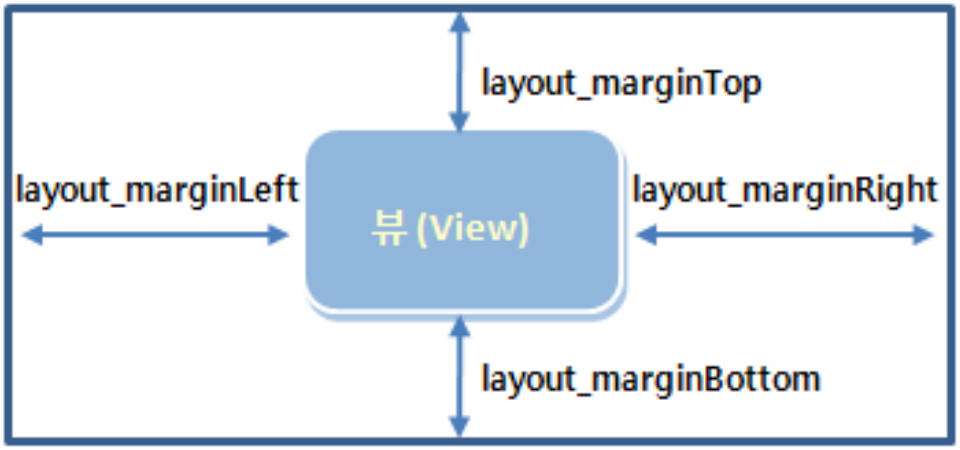
리니어 레이아웃 – 정렬을 위해 사용할 수 있는 값

정렬 속성값	설 명
top	- 대상 객체를 위쪽 끝에 배치하기
bottom	- 대상 객체를 아래쪽 끝에 배치하기
left	- 대상 객체를 왼쪽 끝에 배치하기
right	- 대상 객체를 오른쪽 끝에 배치하기
center_vertical	- 대상 객체를 수직 방향의 중앙에 배치하기
center_horizontal	- 대상 객체를 수평 방향의 중앙에 배치하기
fill_vertical	- 대상 객체를 수직 방향으로 여유 공간만큼 확대하여 채우기
fill_horizontal	- 대상 객체를 수평 방향으로 여유 공간만큼 확대하여 채우기
center	- 대상 객체를 수직 방향과 수평 방향의 중앙에 배치하기
fill	- 대상 객체를 수직 방향과 수평 방향으로 여유 공간만큼 확대하여 채우기
clip_vertical	- 대상 객체의 상하 길이가 여유 공간보다 클 경우에 남는 부분을 잘라내기 - top|clip_vertical 로 설정한 경우 아래쪽에 남는 부분 잘라내기 - bottom|clip_vertical 로 설정한 경우 위쪽에 남는 부분 잘라내기 - center_vertical|clip_vertical 로 설정한 경우 위쪽과 아래쪽에 남는 부분 잘라내기
clip_horizontal	- 대상 객체의 좌우 길이가 여유 공간보다 클 경우에 남는 부분을 잘라내기 - right|clip_horizontal 로 설정한 경우 왼쪽에 남는 부분 잘라내기 - left|clip_horizontal 로 설정한 경우 오른쪽에 남는 부분 잘라내기 - center_horizontal|clip_horizontal 로 설정한 경우 왼쪽과 오른쪽에 남는 부분 잘라내기

리니어 레이아웃 – 마진과 패딩 설정하기



[padding을 이용한 뷰 내부의 여백 주기]

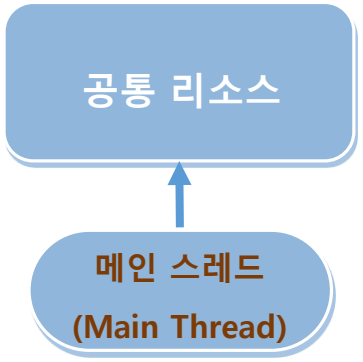


[layout_margin을 이용한 부모 여유공간과의 여백 주기]

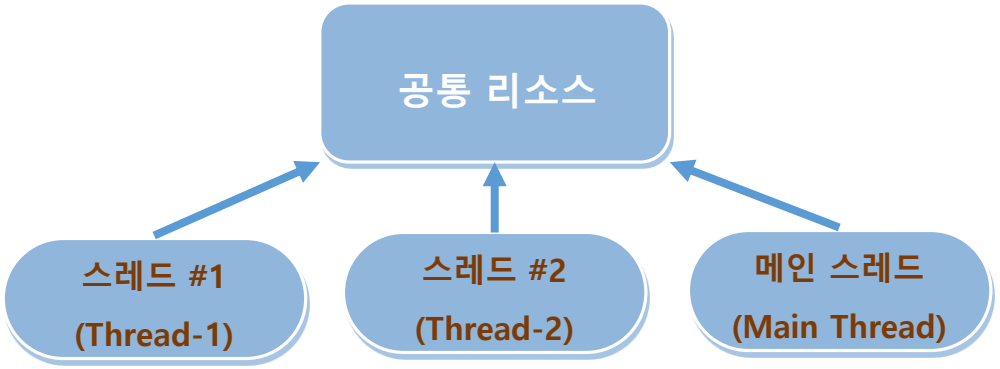
- **padding 속성**
  - 뷰 안의 내용물인 텍스트나 이미지와 뷰 안의 영역 사이의 여백을 줄 수 있는 방법

- **layout_margin 속성**
  - 부모 컨테이너의 여유 공간과 뷰 사이의 여백을 줄 수 있는 방법

멀티 스레드



(1) 프로젝트 생성 시



(2) 별도의 스레드 생성 시

[멀티스레드 시스템에서 시스템에서 공통 메모리 리소스 접근]

- 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 동시 수행되는 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 UI접근가능

핸들러 사용하기

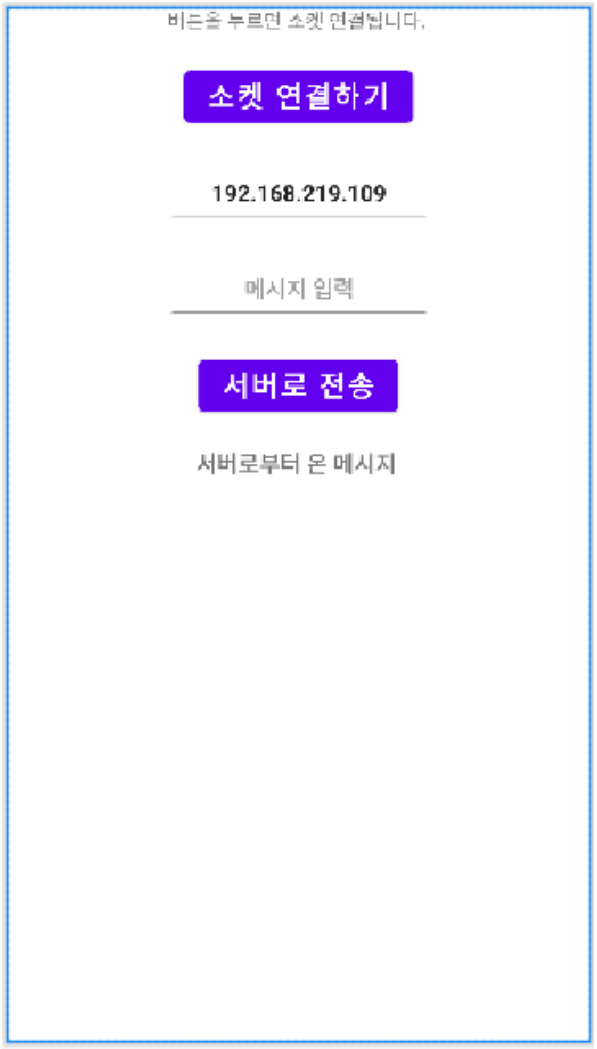
시나리오	설 명
스레드 사용	• 스레드는 동일 프로세스 내에 있기 때문에 작업 수행의 결과를 바로 처리할 수 있음 • 그러나 UI 객체는 직접 접근할 수 없으므로 핸들러(Handler) 객체를 사용함

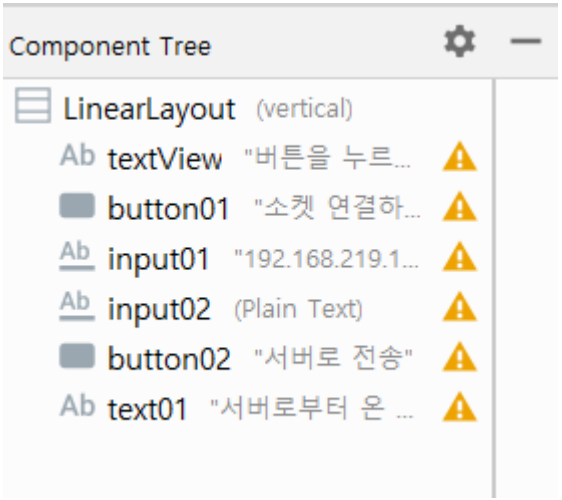
안드로이드 소켓 통신 예제_서버

```
1.  #include <stdio.h>
2.  #include <string.h>
3.  #include <sys/socket.h>
4.  #include <arpa/inet.h>
5.  #include <unistd.h>

6.  #define PORT 9000

7.  int main(void){
8.      int  s_socket, c_socket;
9.      struct sockaddr_in s_addr, c_addr;

10.     int n;
11.     int len;
12.     char rcvBuffer[BUFSIZ];

13.     s_socket = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);

14.     memset(&s_addr, 0, sizeof(s_addr));
15.     s_addr.sin_addr.s_addr = htonl(INADDR_ANY);
16.     s_addr.sin_family = AF_INET;
17.     s_addr.sin_port = htons(PORT);
18.
19.     if(bind(s_socket, (struct sockaddr*)&s_addr, sizeof(s_addr)) == -1){
20.         printf("Can not Bind!!!₩n");
21.         return -1;
22.     }
```

```
22.     if(listen(s_socket, 5) == -1){
23.         printf("Listen Fail!!!₩n");
24.         return -1;
25.     }

26.     printf("Echo Server started...₩n");
27.     while(1){
28.         len = sizeof(c_addr);
29.         c_socket = accept(s_socket, (struct sockaddr*)&c_addr, &len);
30.         printf("Connected IP : %s₩n", inet_ntoa(c_addr.sin_addr));

31.         while((n = read(c_socket, rcvBuffer, sizeof(rcvBuffer))) > 0){
32.             rcvBuffer[n] = '₩0';
33.             printf("%s", rcvBuffer);
34.             write(c_socket, rcvBuffer, n);
35.
36.         }
37.         close(c_socket);
38.     }
39.     close(s_socket);
40.     return 0;
41. }
```

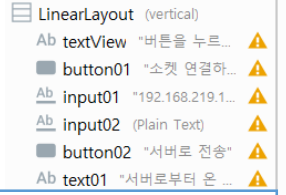
안드로이드 소켓 통신 예제_클라이언트






```
android:id="@+id/button01"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_gravity="center"
android:layout_marginTop="20dp"
android:text="소켓 연결하기"
android:textSize="20sp"
android:textStyle="bold"
```

안드로이드 소켓 통신 예제_클라이언트




```
package com.bong.echo_client;

import…

public class MainActivity extends AppCompatActivity {
    Socket socket;
    OutputStream os;
    InputStream is;
    BufferedReader in ;
    PrintWriter out;

    ConnectThread thread;

    Button button01, button02;
    EditText input01, input02;
    TextView text01;
    String msg;

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

        input01 = (EditText) findViewById(R.id.input01);
        input02 = (EditText) findViewById(R.id.input02);
        text01 = (TextView) findViewById(R.id.text01);
        button01 = (Button) findViewById(R.id.button01);
        button02 = (Button) findViewById(R.id.button02);
        button02.setEnabled(false);
```

```
    button01.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            String addr = input01.getText().toString().trim();

            thread = new ConnectThread(addr);
            thread.start();

            button01.setEnabled(false);
            button02.setEnabled(true);
        }
    });
```

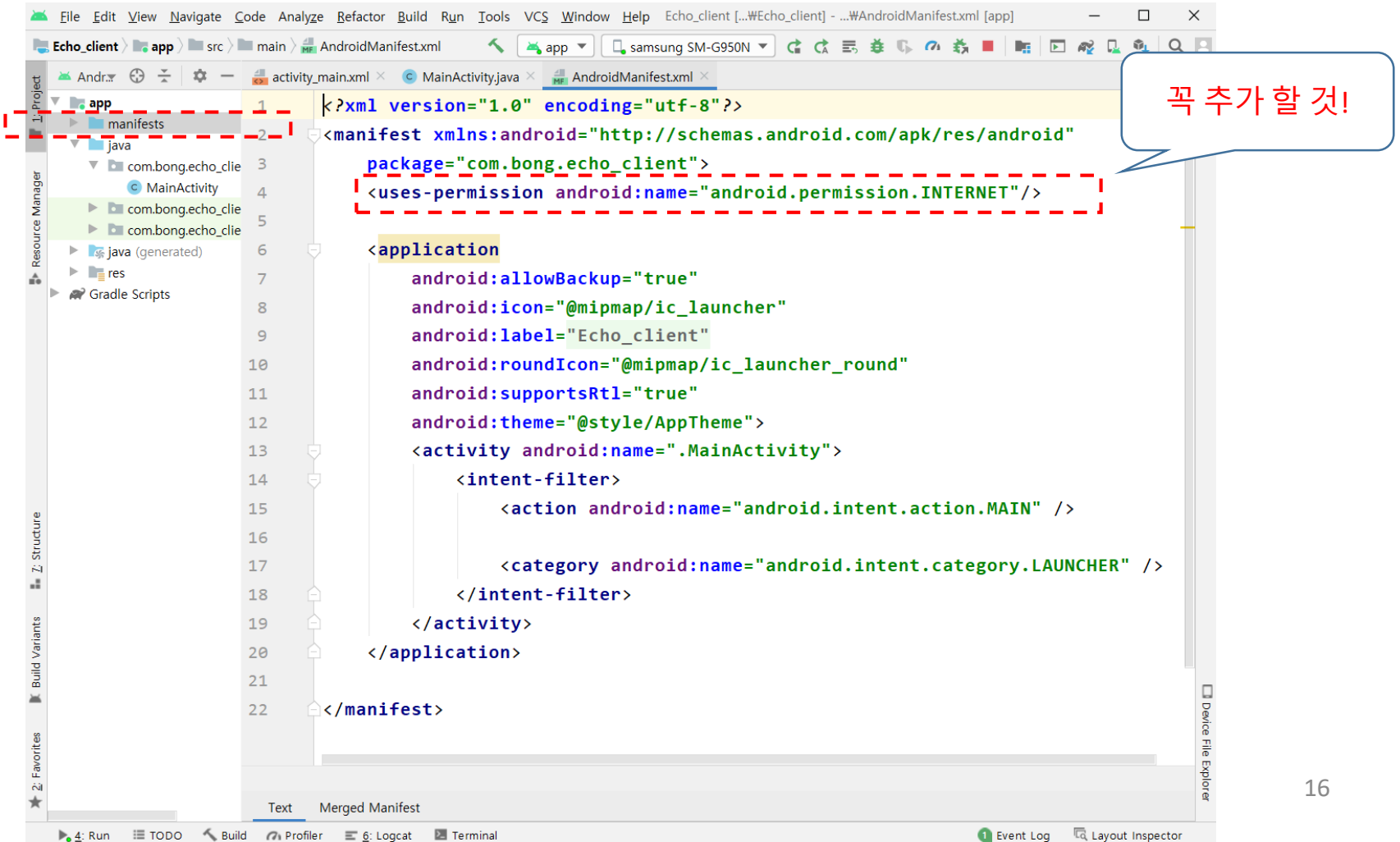
```
    button02.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            msg = input02.getText().toString().trim();

            new Thread(){
                public void run(){
                    out.println(msg);
                    out.flush();
                    thread.readServer();
                }
            }.start();

            if (msg.equals("bye")) {
                thread.setStop();
                button01.setEnabled(true);
                button02.setEnabled(false);
            }

            input02.setText("");
        }
    });
}
```

안드로이드 소켓 통신 예제_클라이언트

```
// * 소켓 연결할 스레드 정의

class ConnectThread extends Thread {

    String hostname;

    public ConnectThread(String addr) {
        hostname = addr;
    }

    public void run() {

        try {
            int port = 9000;

            socket = new Socket(hostname, port);
            os = socket.getOutputStream();
            is = socket.getInputStream();
            in = new BufferedReader(new InputStreamReader(is));
            out = new PrintWriter(os);
        } catch (Exception ex) {
            ex.printStackTrace();
            try {
                socket.close();
            }catch (Exception e){
                e.printStackTrace();
            }
        }

    }

    public void readServer(){
        try {

            String msg1 = in.readLine();
            handler.post(new Runnable() {
                @Override
                public void run() {
                    text01.setText("서버에서 받은 내용 : "+msg1);

                }
            });

        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }

    public void setStop() {
        if(socket.isConnected()) {
            try {
                socket.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

AndroidManifest

- 설치된 앱의 구성요소가 어떤 것인지, 어떤 권한이 부여 되었는지 시스템에게 알려줌



네트워킹 사용 시 주의할 점

- 네트워킹을 사용할 때는 반드시 스레드 사용
  - 최신 버전의 안드로이드에서는 네트워킹을 사용할 때는 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