Lab 3:

Connection of Android

with Wi Fi™

Outline

- 本次Lab課程要在Android上實作以WiFi Direct連線的聊天室。

1. 什麼是Wifi Direct?
2. Android上Wifi Direct的建置
    1. 連線流程
    2. App的架構
    3. 實作WiFi Direct的Activity以及BroadcastReceiver
3. 用Wifi Direct實作聊天室

用到的新物件：
Spinner
ListView

Notice

- 若做到標題為**CHECK POINT**的投影片，請找助教確認進度。
- 程式中有**ToDo**代表要自己完成。

* 本次實驗程式碼較多，請善用
  - auto-fix: alt + Enter
  - auto-complete: control + shift + Enter
  - 右鍵 => generate => override methods, constructors…
  - layout的拖拉介面
  - 其他由Android Studio自動產生的方式

WiFi Direct

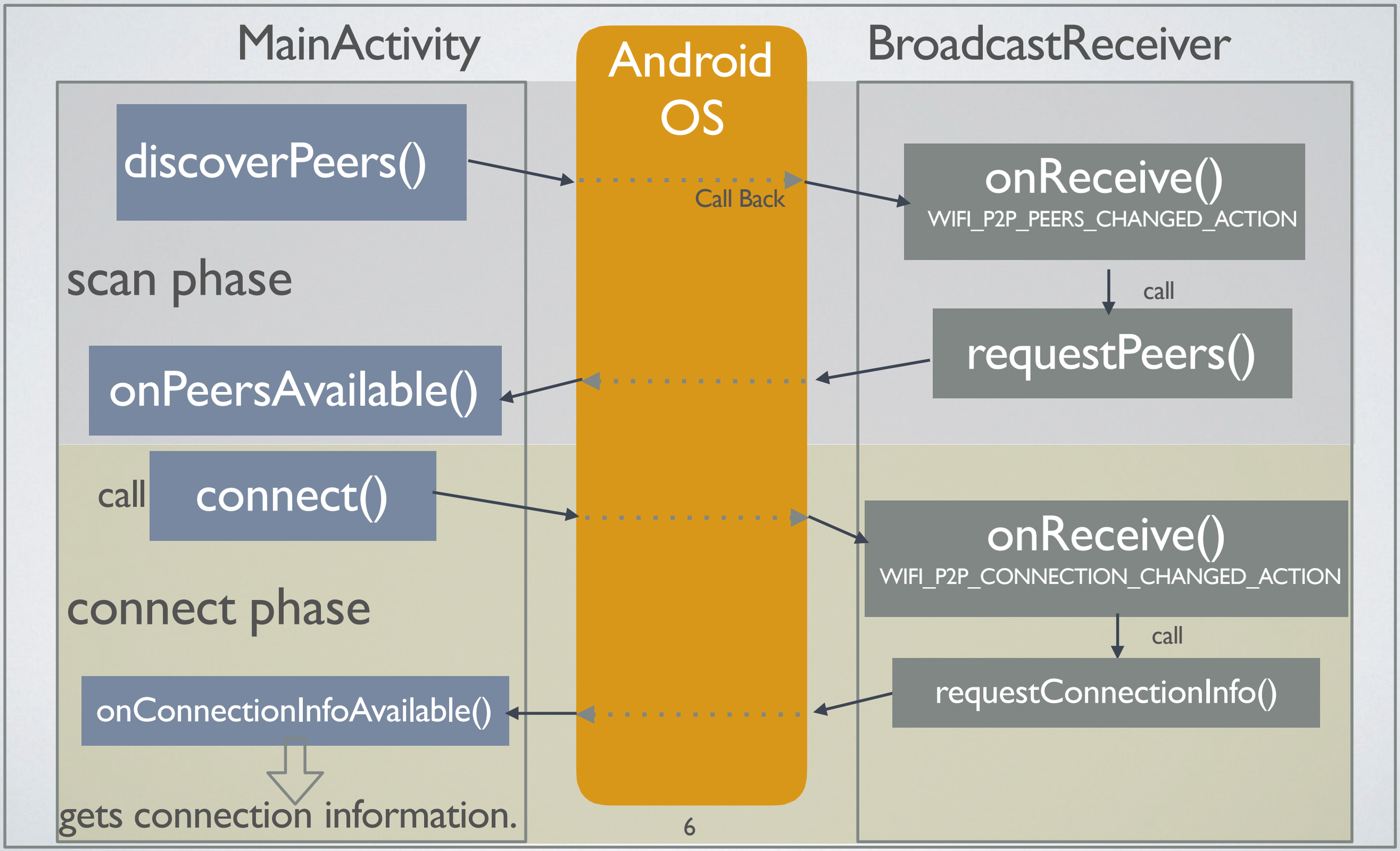
- 由Wifi發展聯盟於2009年制定，以Wifi進行**點對點**連線(不需要經過AP)，進行高速傳輸的一種協定。
- Android 4.0以上支援。
  - 請到Wifi的設定裡找Wifi Direct的選項。
- 裝置進行連線後，會由其中一方作為GroupOwner，雙方都會得到GroupOwner的IP。

Android上WiFi Direct連線的流程

- WiFi Direct的連線步驟大致可以分為兩個部分：

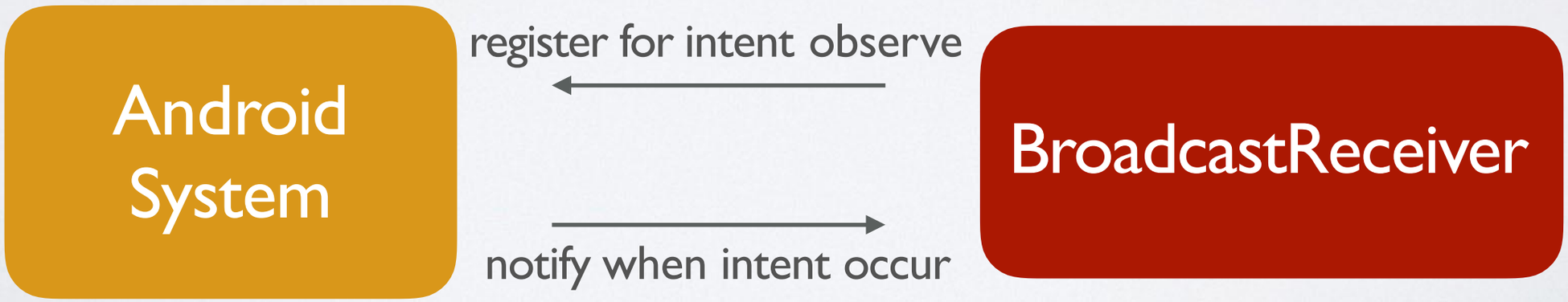
1. SCAN: 掃描附近其他也在執行WiFi Direct程式的裝置
   - 呼叫function請系統執行掃描
   - 呼叫function請系統回傳掃描結果，系統會將結果傳進我們Implement的CallBack function
2. CONNECT: 選取一台裝置的mac執行WiFi Direct連線
   - 呼叫function請系統對特定mac裝置進行連線
   - 呼叫function回傳連線對象的資訊(IP)，系統會將資訊傳進我們Implement的CallBack function

Android上WiFi Direct連線的流程



What's a BroadcastReceiver?

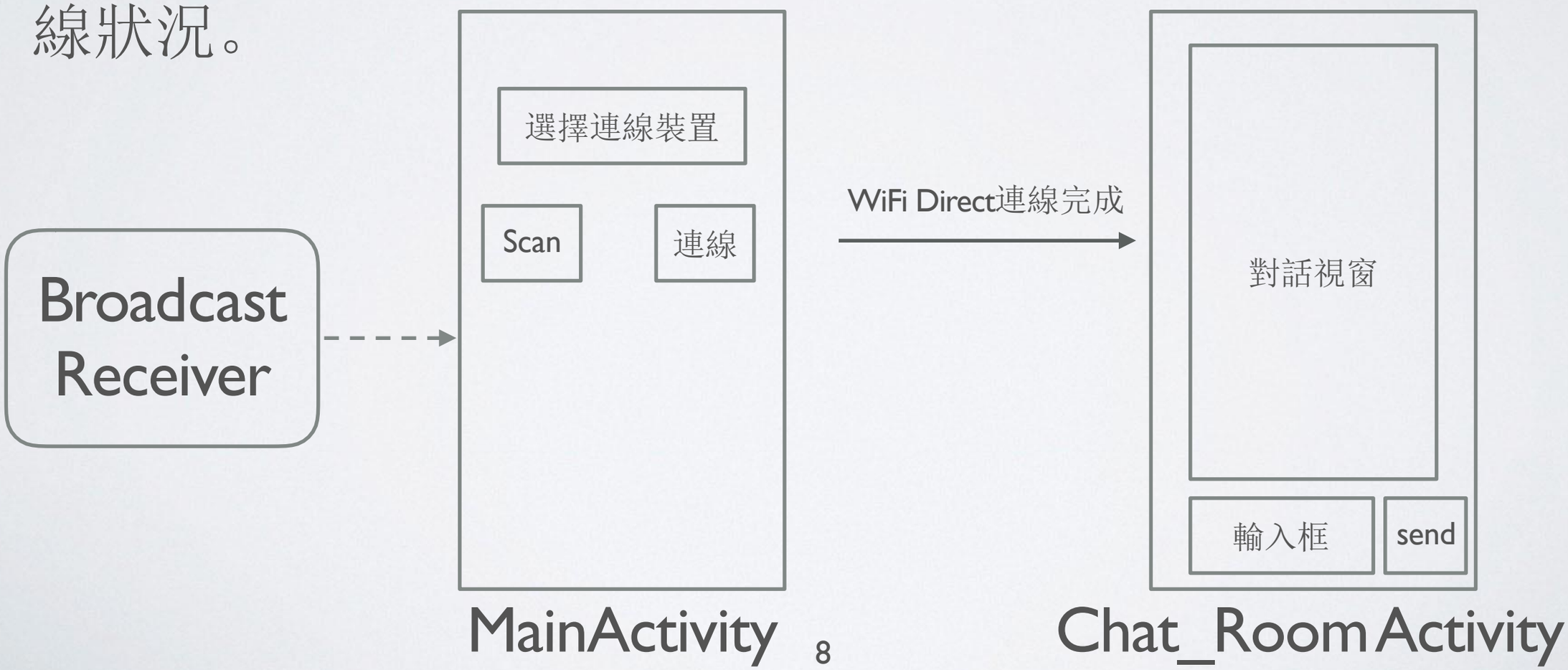
- 用來對系統**註冊(register)**要接收某些由系統所送出的一些訊息(intent)的元件。
- 當系統發出這些intent的時候會提醒BroadcastReceiver。



APP的架構

- 這次實驗要用兩個Activity，一個用來負責WiFi Direct的連線，一個用來做聊天室資料的傳輸，分別各有一個layout。

- 另外還有一個BroadcastReceiver負責在背景監測WiFi Direct的連線狀況。



實作WiFi Direct的Activity

MainActivity

- 在MainActivity上，主要要做的功能為：
  - 尋找其他WiFi Direct的裝置
  - 選擇要連線的對象
  - 連線完成，得到GroupOwner的IP

- 步驟：
  1. 建立Layout
  2. 建立BroadcastReceiver
  3. 請求WifiP2pManager
  4. 裝置搜尋、列出
  5. 送出連線請求，得到IP

note:
本次實驗所做出來的程式是以最簡單的方式達成連線，所以有很多不穩的地方。

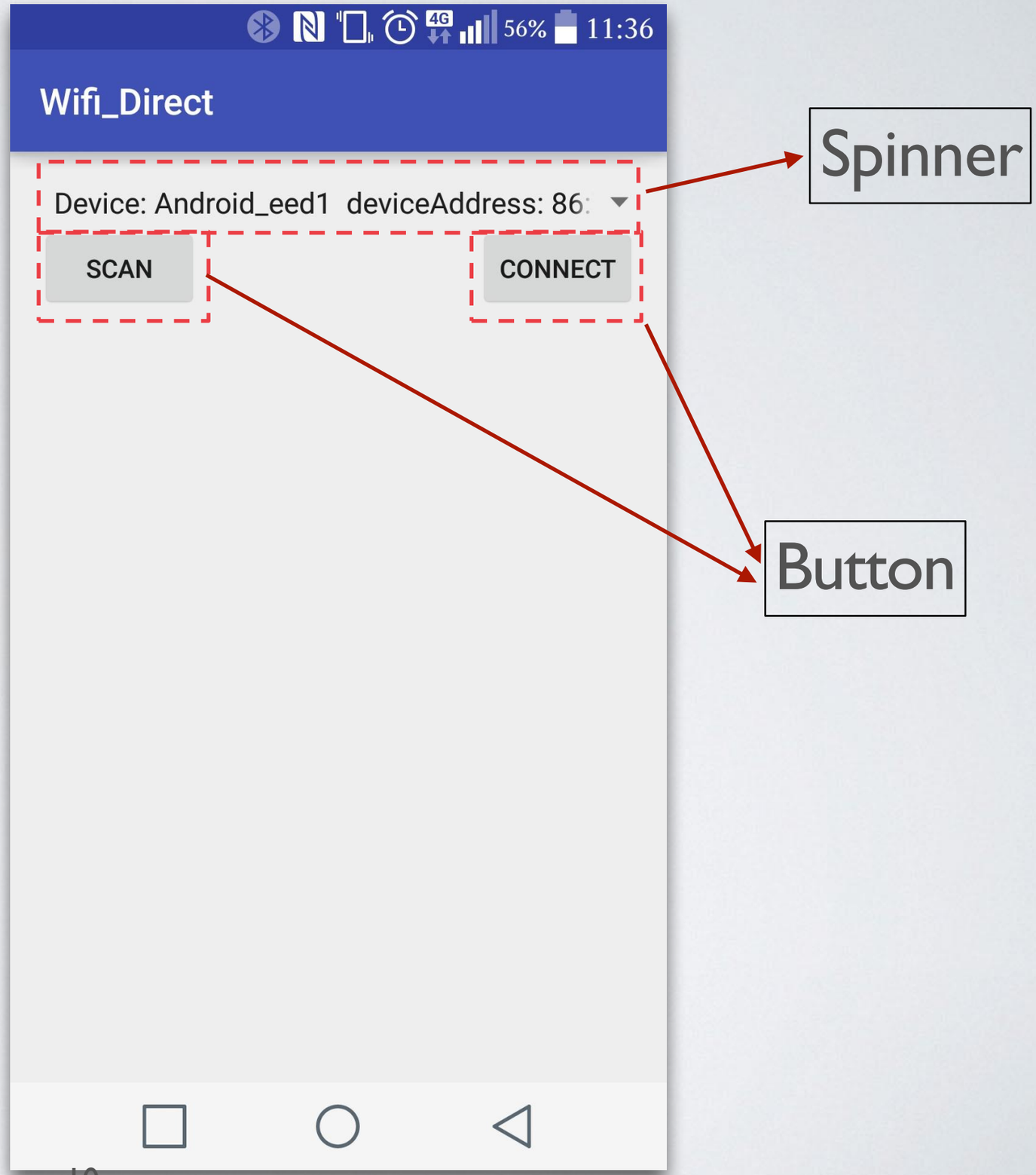
activity_main Layout

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.wifip2p.jengpoyuan.wifi_direct.MainActivity">


    <Spinner
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/spinner"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:spinnerMode="dropdown" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="scan"
        android:id="@+id/scan"
        android:layout_below="@+id/spinner"
        android:layout_alignParentStart="true" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="connect"
        android:id="@+id/connect"
        android:layout_below="@+id/spinner"
        android:layout_alignParentEnd="true" />
</RelativeLayout>
```



不用跟這個一樣，有把三個物件拉出來就可以。

Manifest.xml

- 加入這次App所需要的權限。

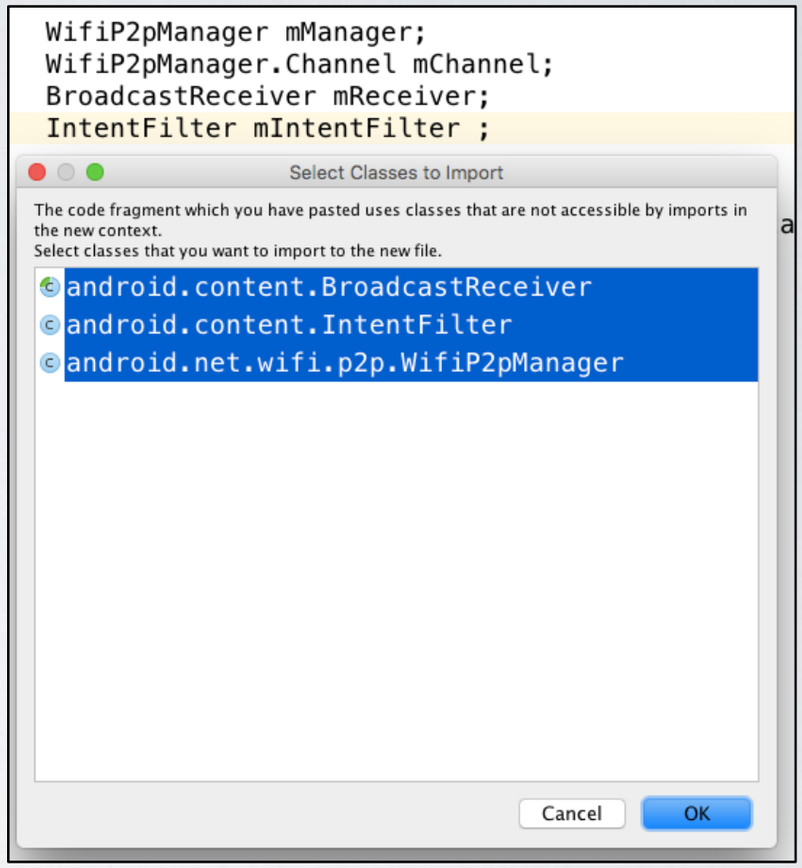
```
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

MainActivity

- Initialize a empty Activity.

- Declare some variables for Wifi p2p.

```
WifiP2pManager mManager;
WifiP2pManager.Channel mChannel;
BroadcastReceiver mReceiver;
IntentFilter mIntentFilter ;
```



自動import要用到的Library

- Go to onCreate()，將Wifi網路斷線

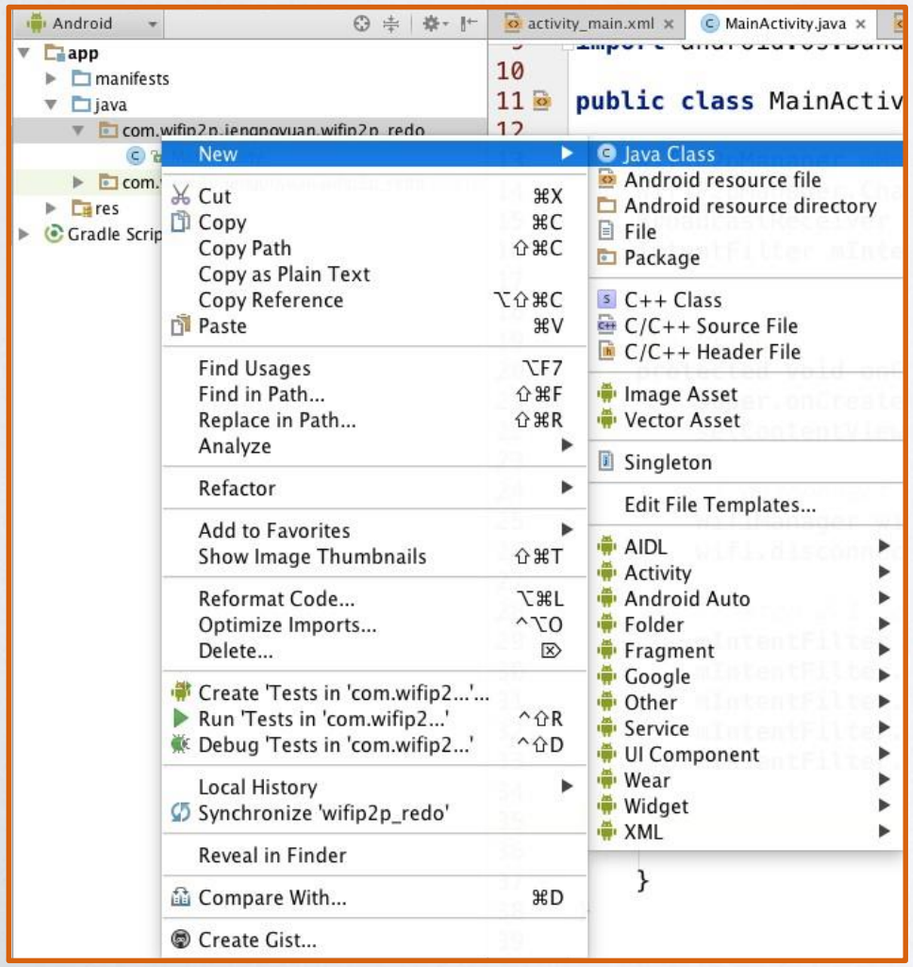
```
// disconnect to wifi AP
WifiManager wifi = (WifiManager) getSystemService(Context.WIFI_SERVICE);
wifi.disconnect();
```

因為使用Wifi Direct時不能同時連到AP，所以要先斷線

- 設定BroadcastReceiver要收的intent。

```
// step 2.1 set intentfilter
mIntentFilter = new IntentFilter() ;
mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION);
mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION);
mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION);
mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION);
```

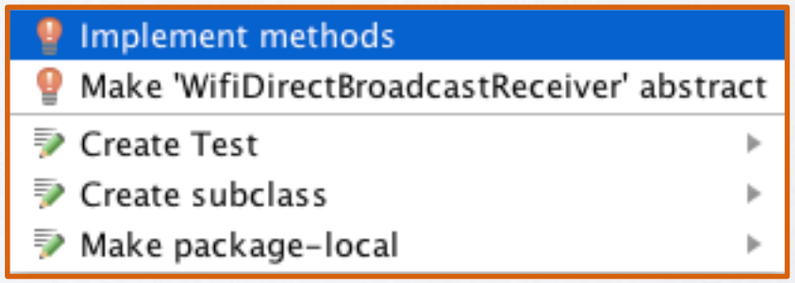
- 新增一個java class，實作BroadcastReceiver
  - name:WifiDirectBroadcastReceiver



- Extend BroadcastReceiver

```
1  package com.wifip2p.jengpoyuan.wifip2p_redo;
2
3  import android.content.BroadcastReceiver;
4
5  public class WifiDirectBroadcastReceiver extends BroadcastReceiver{
6
7  }
```

auto-fix: alt + enter



- Declarations

```
private WifiP2pManager manager;
private WifiP2pManager.Channel channel;
private MainActivity activity;
```

- right click, generate, constructor

```
public WifiDirectBroadcastReceiver(
        WifiP2pManager manager,
        WifiP2pManager.Channel channel,
        MainActivity activity) {
    super() ;
    this.manager = manager;
    this.channel = channel;
    this.activity = activity;
}
```

- 回到mainActivity，初始化WiFi Direct相關物件。

```
mManager = (WifiP2pManager) getSystemService(Context.WIFI_P2P_SERVICE);
mChannel = mManager.initialize(this, getMainLooper(), null);
mReceiver = new WifiDirectBroadcastReceiver(mManager, mChannel, this);
```

- 在onResume()跟onPause()裡註冊Receiver跟他的Filter

```
@Override
protected void onResume() {
    super.onResume();
    registerReceiver(mReceiver, mIntentFilter) ;
}
@Override
protected void onPause() {
    super.onPause();
    unregisterReceiver(mReceiver) ;
}
```

Hint:點右鍵的generate裡都可以自動產生這些可以override的function

- 目前為止，App已經可以監控Wifi Direct的連線狀態，但我們還沒實作監控到不同狀況要做什麼事，以及尋找其他裝置、請求連線...等功能。

- 接下來要開始實作操作Wifi Direct的功能。

- 初始化layout中的views。
  - Declarations:

```
Button scan, connect ;
Spinner device_spinner ;
ArrayAdapter<String> mArrayAdapter ;
ArrayList<String> device_list ;
```

控制spinner中內容的物件

  - FindViewById:

```
scan = (Button) findViewById(R.id.scan) ;
connect = (Button) findViewById(R.id.connect) ;
device_spinner = (Spinner) findViewById(R.id.spinner) ;
```

- 設定尋找裝置的按鈕(scan)的作用

```
scan.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        mManager.discoverPeers(mChannel, new WifiP2pManager.ActionListener() {
            @Override
            public void onSuccess() {
                Log.d("notice", "onSuc") ;// Log.d show the log on Android Studio
            }

            @Override
            public void onFailure(int reason) {
                Log.d("notice","onFail") ;
            }
        });
    }
});
```

一定要用一個Action Listener恩function有沒有呼叫成功

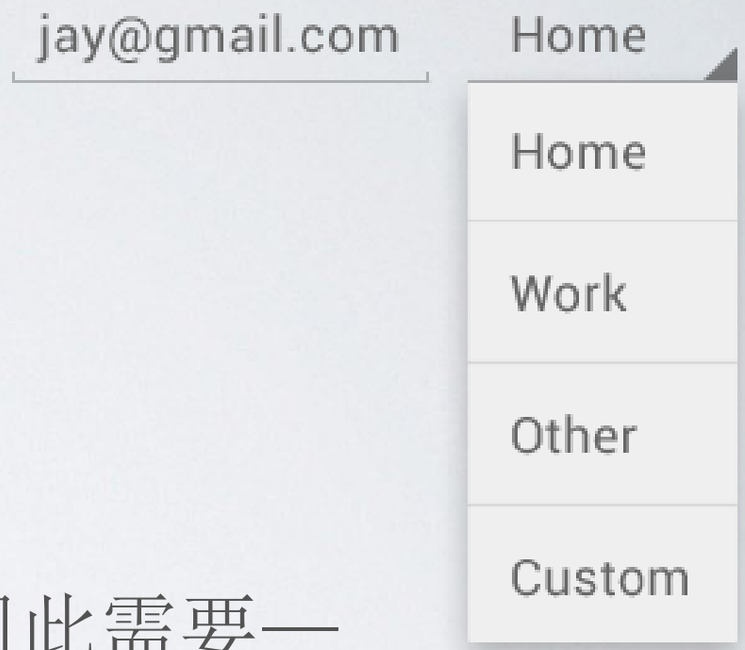
Log.d可以將文字顯示在Android Studio上，可用來DeBug。不影響程式執行。

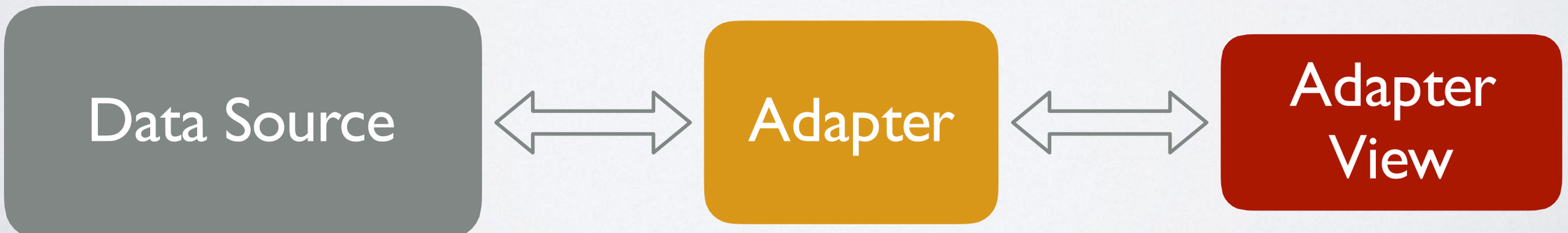
Hint:
善用Android Studio的自動完成

- 按下時呼叫discoverPeers，讓系統去掃描其他的Wifi p2p裝置。系統得到結果後會通知BroadcastReceiver，之後會實作被通知以後要執行的事情。

Spinner

- 下拉式選單物件。
- 需要一個List來裝要顯示的內容。
  - 可以寫死在values.strings裡面
  - 我們需要可以動態更改內容的Spinner，因此需要一個ArrayAdapter負責管理內容。





- ArrayList
- ArrayAdapter
- Spinner
- ListView

- 設定ArrayAdapter:

```
// step 5.3: set arrayAdapter
device_list = new ArrayList<>() ;
mArrayAdapter =
        new ArrayAdapter<>(this, android.R.layout.simple_spinner_item, device_list) ;

mArrayAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
// step 5.4: set spinner
device_spinner.setAdapter(mArrayAdapter);
```

簡易的Spinner形式　　要顯示的List

設定下拉式選單(Spinner)的形式

設定Spinner所使用的Adapter

- 到此為止，Spinner所顯示的是一個空的選單，我們接下來要將呼叫DiscoverPeers()後所得到的裝置放入選單當中。
  - note: 系統掃描到其他裝置後，會送出一個**WIFI_P2P_PEERS_CHANGED_ACTION**的**intent**到BroadcastReceiver裡的onReceive()。

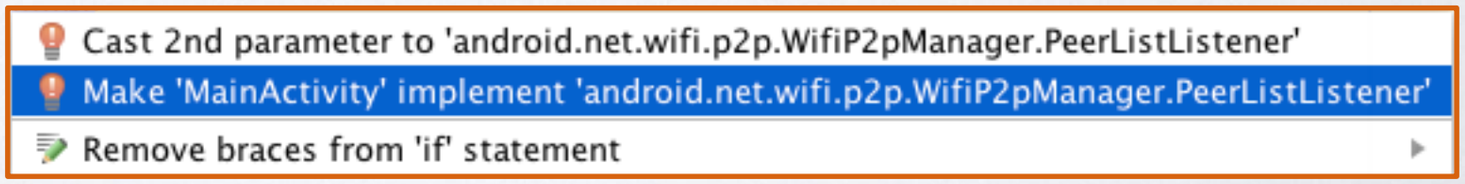
- BroadcastReceiver裡收到intent後把它的action取出

```
String action = intent.getAction();
```

- 再用if/else的方式比對得到的action是不是WIFI_P2P_PEERS_CHANGED_ACTION，因為系統只有告知裝置清單**有改變**，要詳細資料還需請求manager提供peers的資訊。

```
String action = intent.getAction();
if (WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action)) {
    if (manager != null) {
        manager.requestPeers(channel, activity);
    }
}
```

- 此時會看到第二個參數是錯的，因為我們還沒有在MainActivity中實作請求後的callback function接收peers資訊。
  - auto-fix選擇實作



- 在MainActivity中自動產生requestPeers的callback function後，先宣告一個存放結果的變數。

```
private ArrayList<WifiP2pDevice> peer_list = new ArrayList<WifiP2pDevice>();
```

- 到onPeersAvailable()裡，把Devices的資料存入peer_list
- 再更新Spinner的ArrayAdapter裡的內容

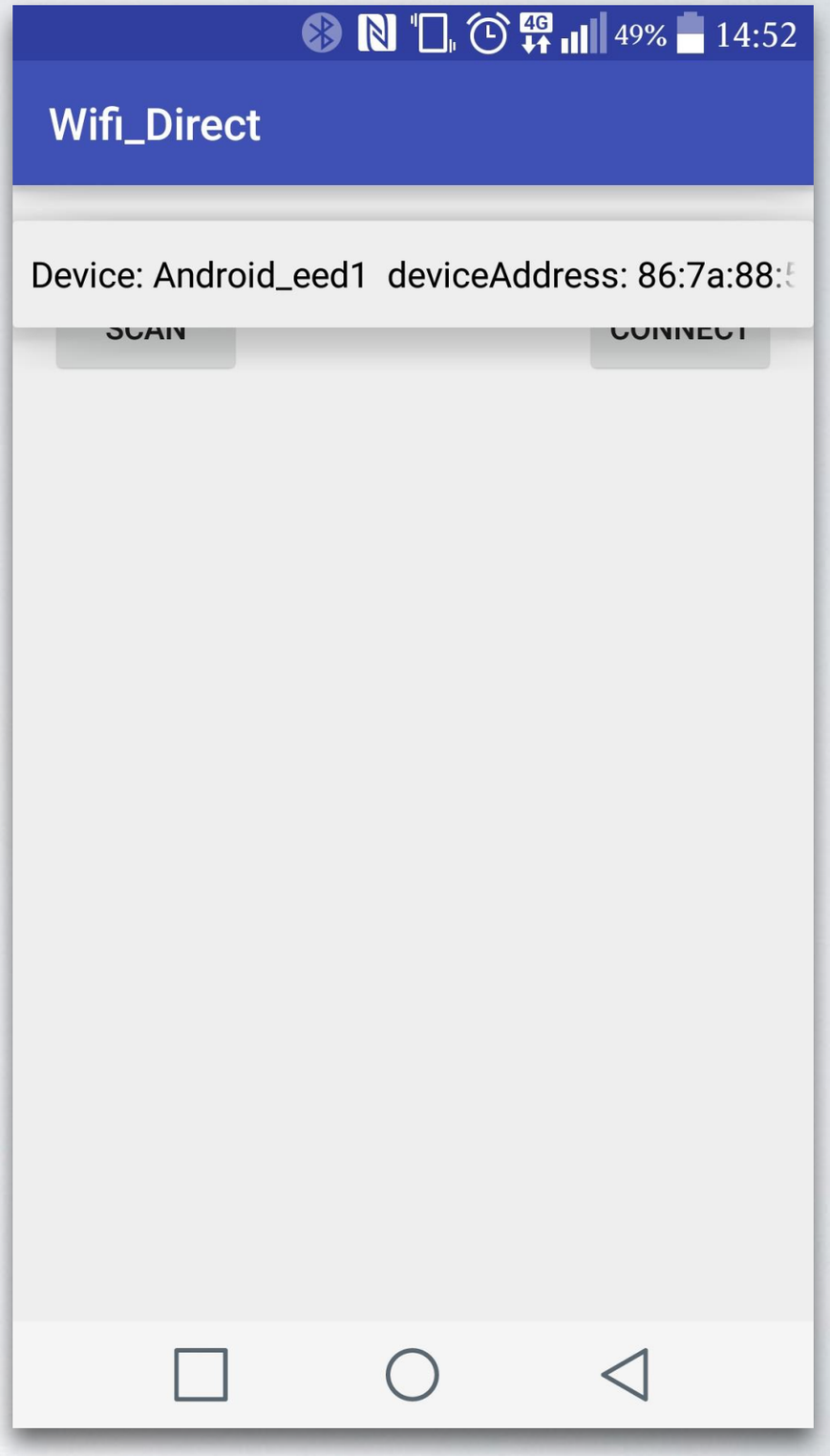
```
@Override
public void onPeersAvailable(WifiP2pDeviceList peers) {
    Log.d("onPeersAvailable","peers") ;
    // for connection config
    peer_list.clear();
    peer_list.addAll(peers.getDeviceList());
    // for spinner content
    for(WifiP2pDevice d:peers.getDeviceList()){
        device_list.add(d.toString()) ;
    }
    mArrayAdapter.notifyDataSetChanged();
}
```

notifyDataSetChanged()會叫**adapter**通知它的view內容有更新

- 此時，按下scan按鈕，就可以看到其他裝置了。

CHECK POINT

- 按下scan按鈕，可以看到其他Wifi Direct的裝置。
  - **注意：**請開啟Wifi以及Wifi Direct



- 得到可連線的裝置後，我們要實作connect按鈕的功能
  - 抓出由Spinner所選到的裝置
  - 取出它的mac address，呼叫connect()

```
connect.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        int idx = device_spinner.getSelectedItemPosition() ;
        WifiP2pDevice device = peer_list.get(idx) ;
        WifiP2pConfig config = new WifiP2pConfig();
        config.deviceAddress = device.deviceAddress;

        mManager.connect(mChannel, config, new WifiP2pManager.ActionListener() {
            @Override
            public void onSuccess() {

            }

            @Override
            public void onFailure(int reason) {

            }
        });
    }
});
```

從spinner抓選取的位置

由位置從peer_list中取device

將其設定檔放入config物件

呼叫manager的connect()

呼叫connect()後，系統會在**連線成功後**送出
WIFI_P2P_CONNECTION_CHANGED_ACTION的intent給BroadcastReceiver

- 到BroadcastReceiver寫收到WIFI_P2P_CONNECTION_CHANGED_ACTION 的intent時要做的事

```
else if(WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION.equals(action)) {
    NetworkInfo networkInfo = (NetworkInfo) intent
            .getParcelableExtra(WifiP2pManager.EXTRA_NETWORK_INFO);
    Log.d("connection change", "connect") ; // debug
    if (networkInfo.isConnected()) {
        manager.requestConnectionInfo(channel, activity);
    }
}
```

取得現在的網路資訊

判斷如果已經連線成功，就去請求已經連線的資訊

- 這時會看到第二個參數是紅字，因為還沒實作requestConnectionInfo()的callback function
  - auto-fix => implement

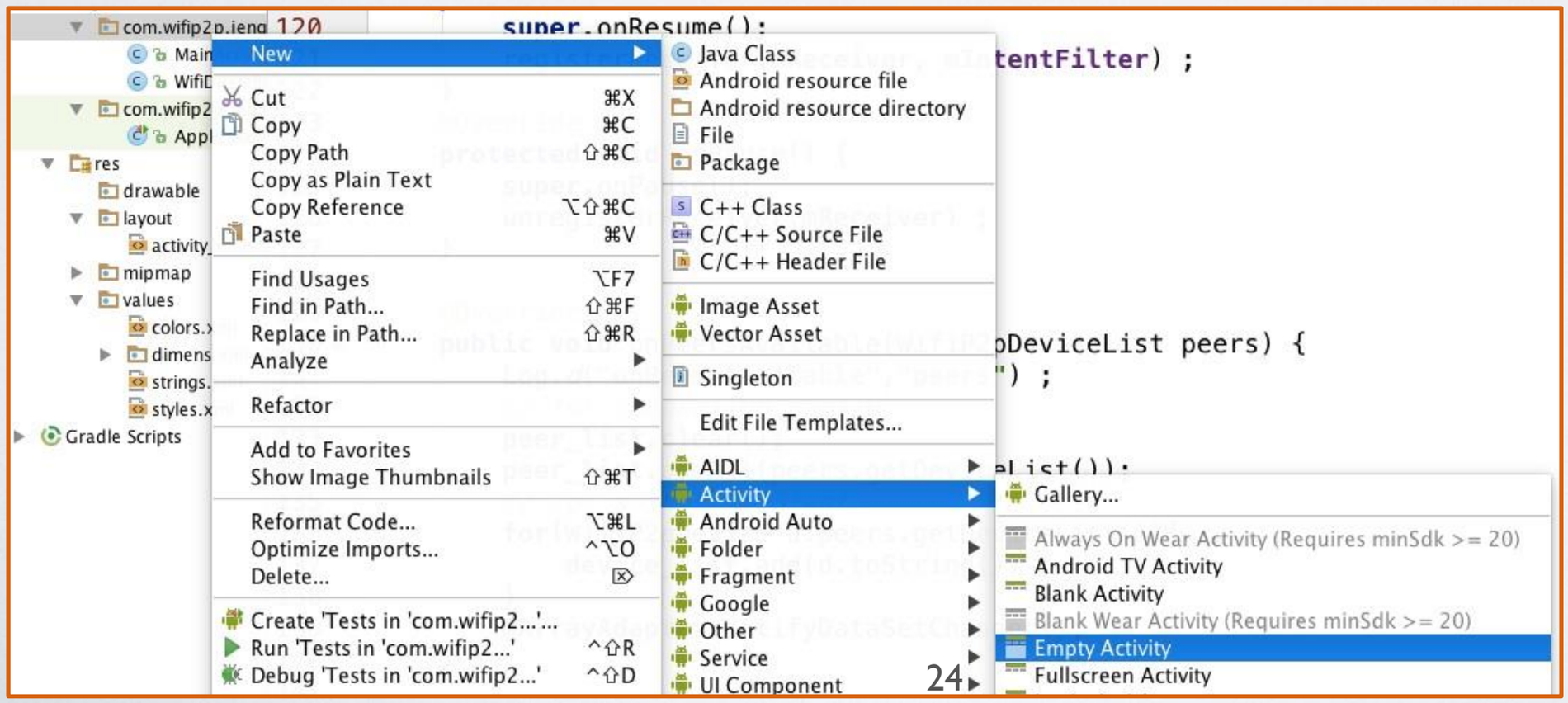
```
@Override
public void onConnectionInfoAvailable(WifiP2pInfo info) {

}
```

- 系統在callback onConnectionInfoAvailable()的時候會傳入一個WifiP2pInfo物件，我們先從中把GroupOwner的IP抓出來。

```
String host_ip = info.groupOwnerAddress.getHostAddress() ;
```

- 這時已經具備socket programing所需的資料後，該是用另一個Activity: ChatRoom來負責聊天室的時候。



- 在得到GroupOwner的IP後，我們統一以**不是GroupOwner**的一方當Client對GroupOwner(server)進行socket連線。 role = client, server.
- 將所需資料傳遞到另一個Activity，並且開啟
  - new一個intent，將要啟動的activity還有IP跟role放進去，

startActivity()可以啟動並將資料帶過去。

```
@Override
public void onConnectionInfoAvailable(WifiP2pInfo info) {
    String host_ip = info.groupOwnerAddress.getHostAddress() ;
    Intent mIntent = new Intent(this, ChatRoom.class) ;
    mIntent.putExtra("IP", host_ip ) ;
    if(info.isGroupOwner){
        mIntent.putExtra("role", "server") ;
    }
    else{
        mIntent.putExtra("role", "client") ;
    }
    startActivity(mIntent); // start a new Activity
}
```

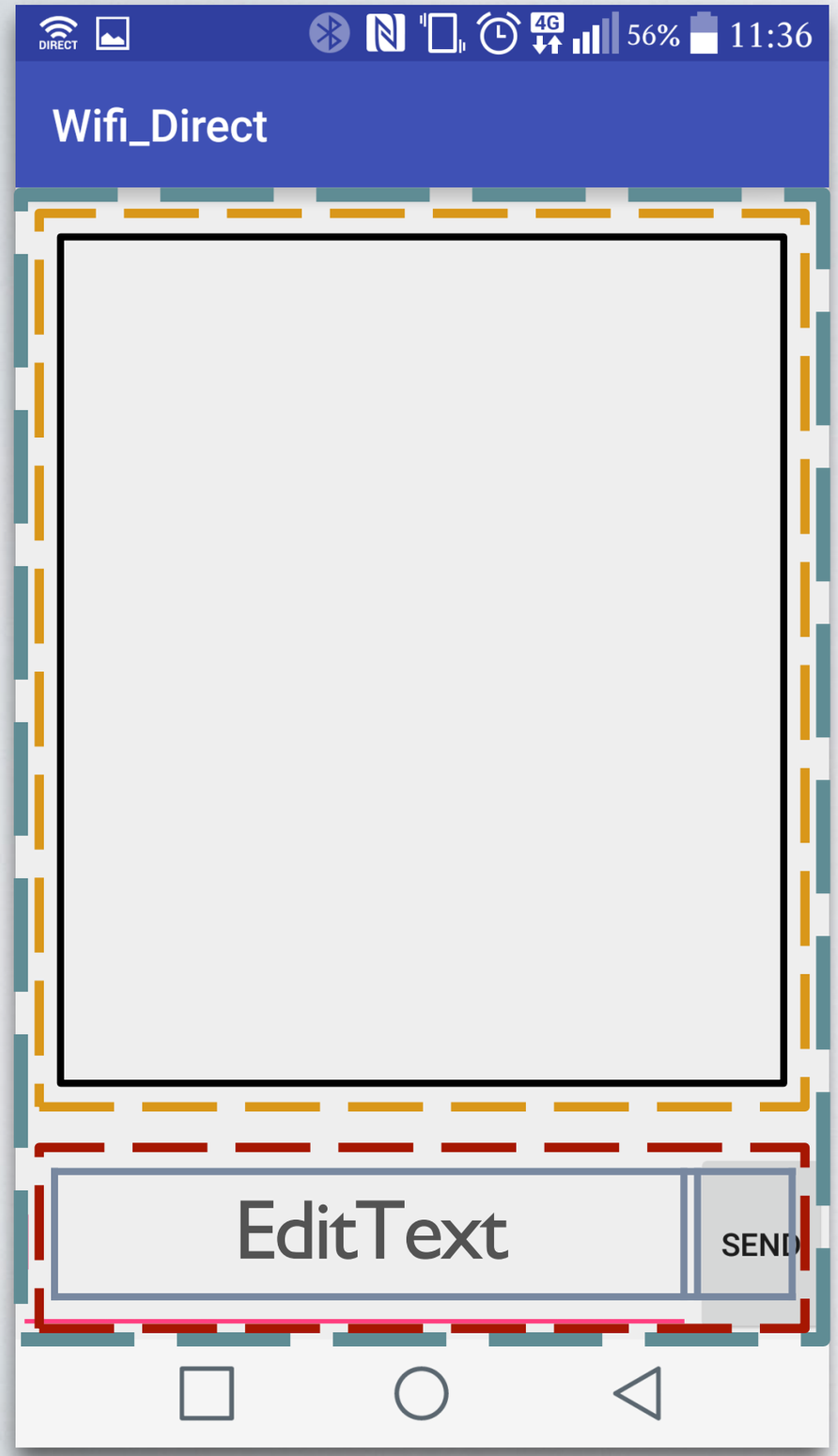
用Wifi Direct實作聊天室

ChatRoom

- ChatRoom Activity中，主要要做的功能為:
  - 建立Socket連線
  - 接收、傳送字串
  - 更新對話框中的文字

- 步驟:
  1. 建立Layout
  2. 建立socket連線
  3. 實作UI介面的功能來顯示、傳送訊息
  4. 建立常駐thread來接收訊息

activity_chat_room layout

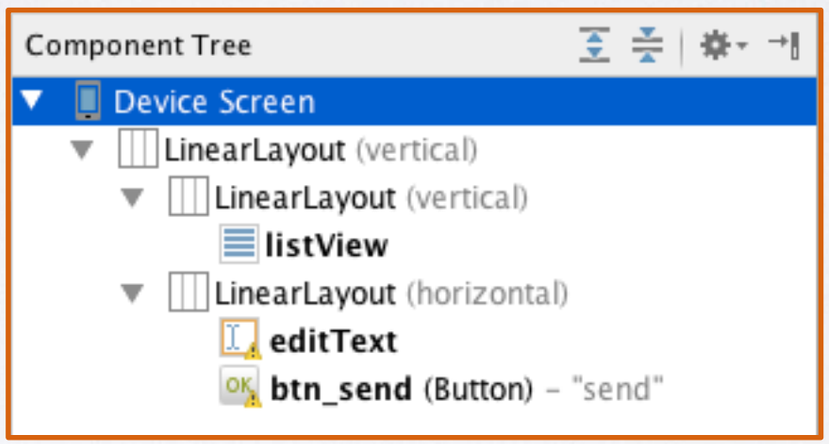


linear layout: vertical

linear layout: vertical

linear layout: horizontal

ListView



button

可以設定物件的android:layout_weight屬性來控制畫面比例

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">


    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="fill_parent"
        android:layout_weight="1">

        <ListView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/listView" />
    </LinearLayout>

    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="fill_parent"
        android:layout_weight="5">

        <EditText
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:id="@+id/editText"
            android:layout_weight="1" />

        <Button
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:text="send"
            android:id="@+id/btn_send"
            android:layout_weight="5" />
    </LinearLayout>
</LinearLayout>
```

ChatRoom

- 首先先宣告socket連線所需變數

```
String ServerAddr, role ;
int port = 8888 ;
Thread receiver ;
```

- 在onCreate()裡用getIntent()取出上一個Activity送過來的資料(role, IP)

```
Intent it = getIntent() ;
role = it.getStringExtra("role") ;
ServerAddr = it.getStringExtra("IP") ;
```

- 由於Android上的Main thread不能執行超過五秒的事情，不然會當掉，所以我們需要另外開一個thread **執行socket的連線與等待對方所送的資料**。
- 到onResume()裡面，把Thread實現並開始，因為還沒有實作Updating_Thread這個class所以是紅字。

```
@Override
protected void onResume() {
    super.onResume();
    // step 10 : set up connection socket
    receiver = new Updating_Thread() ;
    receiver.start();
}
```

- 再宣告一些讀寫用的物件

```
ServerSocket mServerSocket ;
Socket mClientSocket ;
BufferedReader reader ;
BufferedWriter writer ;
Boolean THREAD_CLOSE = false ;
```

- 由於socket programing不在這次實驗的範圍內，所以我們直接提供這個class。

- mHandler會有紅字出現，在下一頁會宣告。

```
class Updating_Thread extends Thread {
    @Override
    public void run() {
        super.run();
        try {
            if(role.equals("client")){
                mClientSocket = new Socket(ServerAddr,port) ;
            }
            else if(role.equals("server")){
                mServerSocket = new ServerSocket(port) ;
                mClientSocket = new Socket() ;
                mClientSocket = mServerSocket.accept() ;
            }
            else {
                Log.d("ERROR", "connection errors") ;
            }
            reader = new BufferedReader(
                    new InputStreamReader(mClientSocket.getInputStream())) ;
            writer = new BufferedWriter(
                    new OutputStreamWriter(mClientSocket.getOutputStream())) ;
            while(!THREAD_CLOSE) {
                String s ;
                s = reader.readLine() ;
                Message m = new Message() ;
                Bundle b = new Bundle() ;
                b.putString("recv_data", s);
                m.setData(b);
                mHandler.sendMessage(m) ;
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

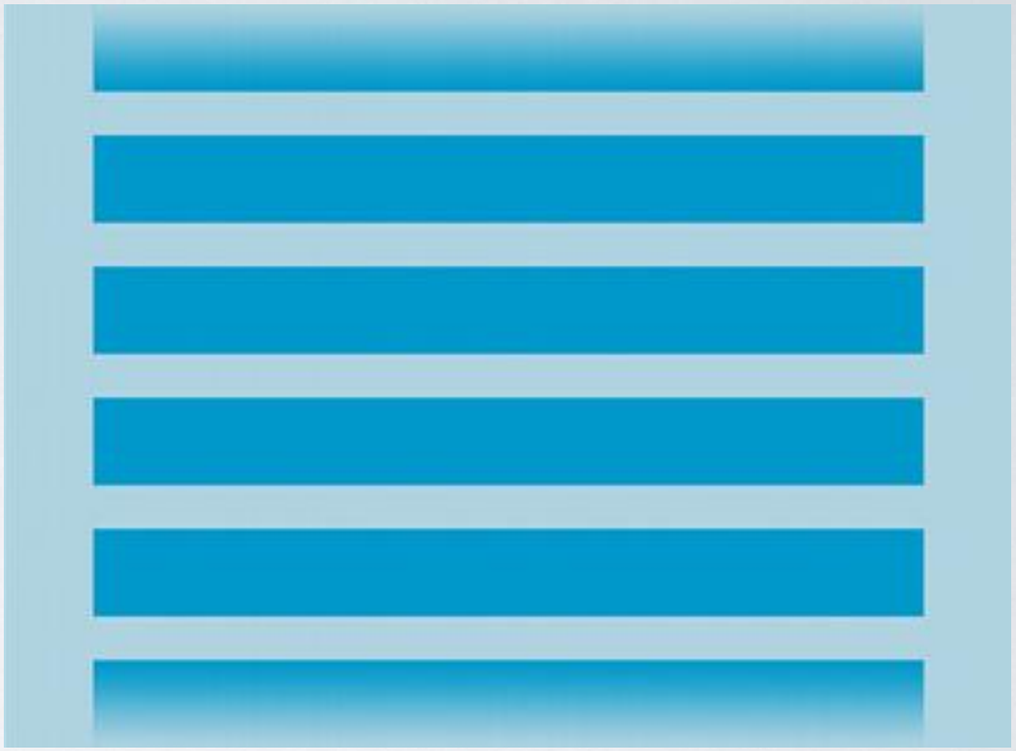
- 接下來開始實作UI介面的控制，首先先宣告物件

```
Button send ;
ListView dialogues ;
EditText input_msg ;
ArrayAdapter<String> mArrayAdapter ;
List<String> msg_data ;
Handler mHandler ;
```

- 其中，我們用ListView來當作聊天視窗顯示的方式
  用EditText提供使用者輸入字串的地方

ListView

- 以條列的方式顯示清單(list)的物件，跟Spinner很接近。
- 也需要一個Adapter來管理要顯示的內容。



- 將ListView dialogues與它的adapter設定好
  - note:不用設定下拉式選單樣式
- 我們要將thread中接收到的訊息更新到dialogues，但在Android系統上不允許其他Thread更改Main Thread的介面，所以我們要在Activity上用Handler來接收Updating_Thread傳回來的訊息，並**更新UI(也就是對話視窗)**。

```
mHandler = new Handler(){
    @Override
    public void handleMessage(Message msg) {
        super.handleMessage(msg);
        String s ;
        s = msg.getData().getString("recv_data") ;
        //Todo.....
    }
} ;
```

s是對方所傳的訊息，這裡需要加入Adapter的List上，並通知更新

- 完成接收端後，剩下發送端要做了。首先，找出**輸入格**跟**發送鈕**兩個物件。

- 設定發送鈕的onClickListener，使他從EditText讀入字串，然後從output stream送出去。

```
String msg ;
msg = input_msg.getText().toString() ;
if(!msg.equals("")){
    try {
        // send a msg
        writer.write(msg + "\n") ;
        writer.flush();

        // ToDo 清空輸入格&將msg加入ListView上

    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

CHECK POINT: DEMO

- 完成一支兩隻手機可以用Wifi Direct互相傳文字的APP。
  - 只需要傳出去在另一支有顯示就可以
- note: 由於程式過於簡易，很多crash的狀況沒處理好，若發生請殺掉程式並重開Wifi就可以。