CSE2MAD LAB 5 – BLUETOOTH SCANNER WITH ALERT DIALOG

AIMS

Today will be making a Bluetooth Scanner app we will be exploring;

- The BluetoothAdapter class
- Dangerous permissions at Runtime
- Alert Dialogs

We will be revisiting various concepts learned through the semester including;

- Broadcast receivers
- ArrayAdapters

STEP 1 – MAKING THE MAIN ACTIVITY UI & REQUESTING PERMISSIONS

a) Create a new project & choose the empty activity template. (See week 1 lab)
b) Create the UI in the visual DESIGN view for the main activity as in Figure 1. If you need a hint see the component tree in Figure 2 and layout code from previous labs. You may get errors, just accept new minimum spinner height and ignore anything else.

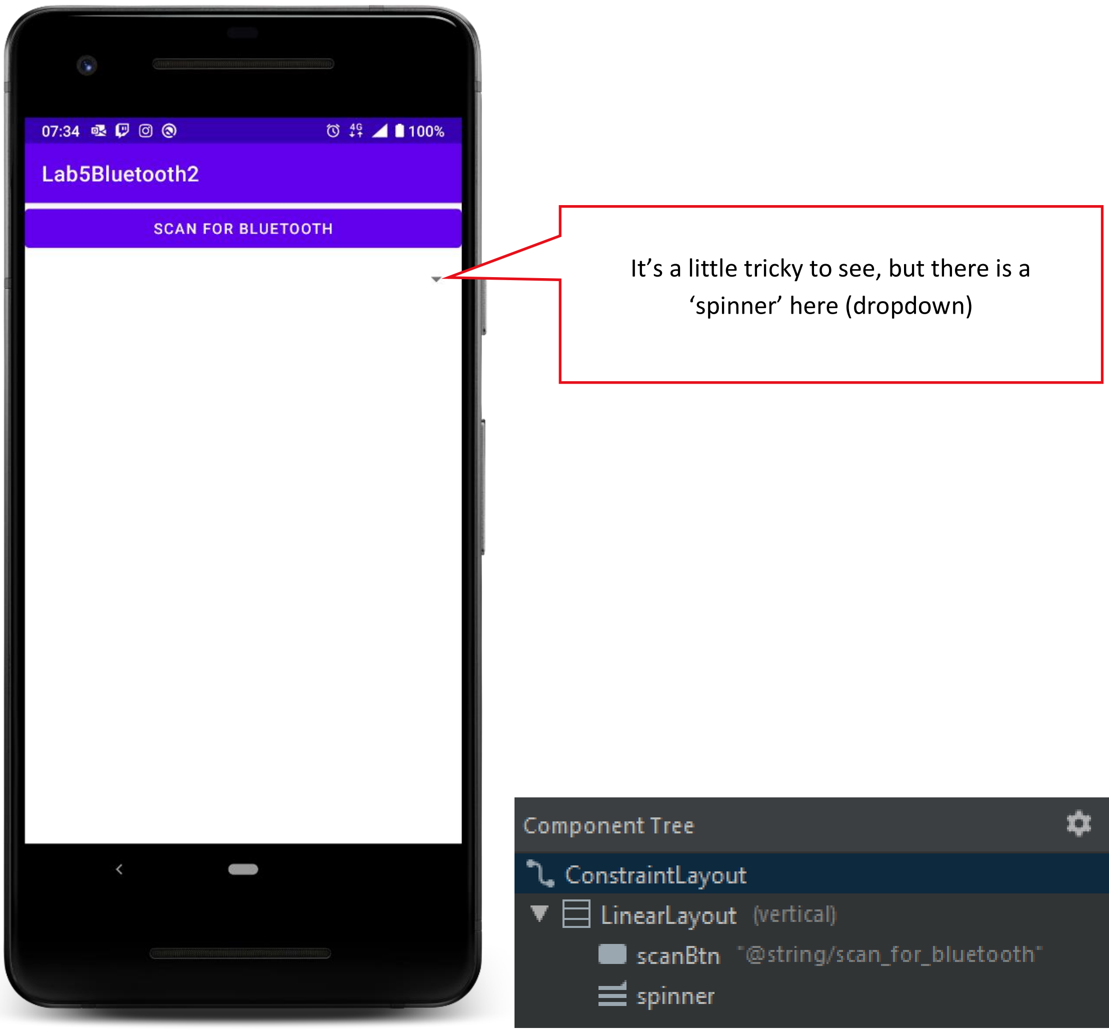
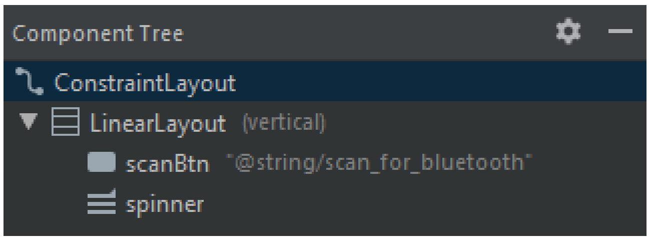

07:34
100%
Lab5Bluetooth2
SCAN FOR BLUETOOTH
It's a little tricky to see, but there is a 'spinner' here (dropdown)

Fig 1



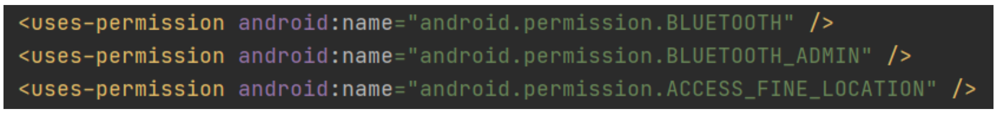
ConstraintLayout
LinearLayout (vertical)
scanBtn "@string/scan_for_bluetooth"
spinner

Fig 2

c) Now set up the member variables for your button & spinner and binding them to your view objects, and add the button onClick Listener

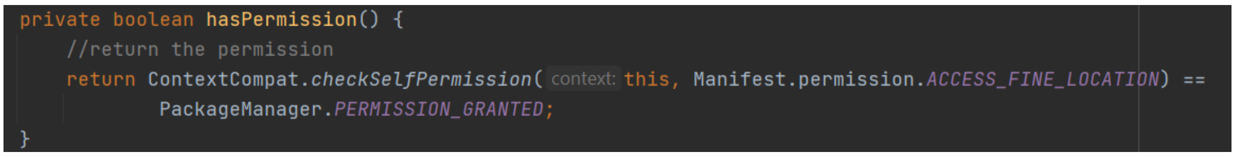
```
private Spinner resultsSpinner;
private Button scanBtn;
private String TAG = "MainAct";

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

    //Spinner
    resultsSpinner = (Spinner) findViewById(R.id.spinner);

    //Button
    scanBtn = (Button) findViewById(R.id.scanBtn);
    scanBtn.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            //Do something on button click

        }
    });
}
```

d) Add our permissions required to use Bluetooth to the normal place, the AndroidManifest.xml just above the application tag.



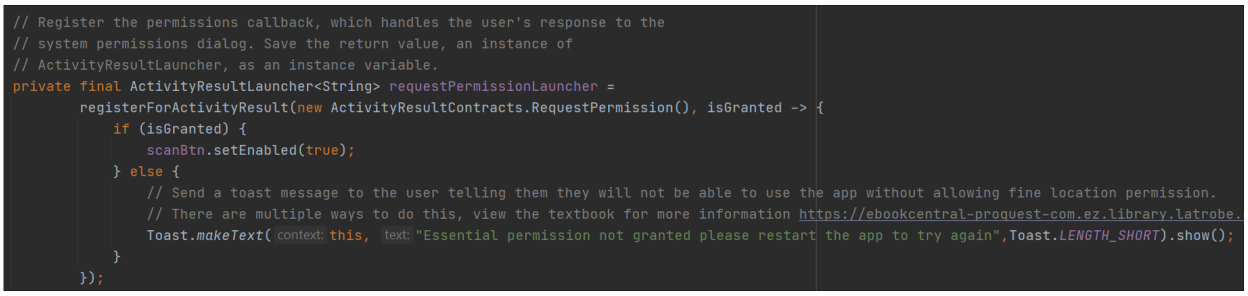
```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

e) Now we need to check permissions. As per this weeks lecture, ACCESS_FINE_LOCATION is considered a 'Dangerous permission" and a request must be made to the user DURING RUNTIME. This is the same process we used in Lab 2 except this time we are just requesting one permission instead of an array of permissions. Make a new class under the onCreate called hasPermission();



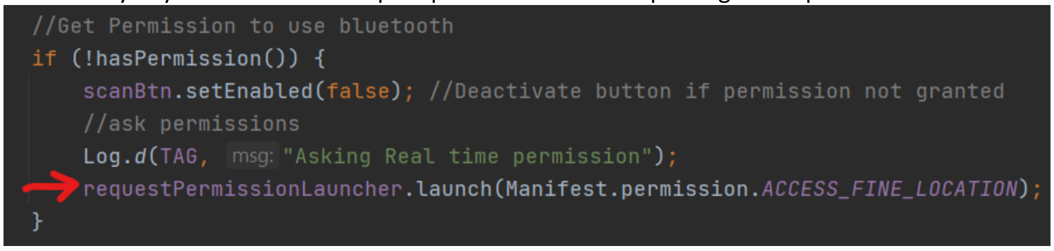
```
private boolean hasPermission() {
    //return the permission
    return ContextCompat.checkSelfPermission( context: this, Manifest.permission.ACCESS_FINE_LOCATION) ==
            PackageManager.PERMISSION_GRANTED;
}
```

f) So now we know if the app already has required permissions but what if it doesn't? Well then we wil need to ask permissions 😉 Create the ActivityResultLauncher outside your onCreate(), this is a callback method that handles the results of the permissions query. It is the new recommended way to request permissions. For more info view https://developer.android.com/training/permissions/requesting If you have troubles reading the image below please use this gist.



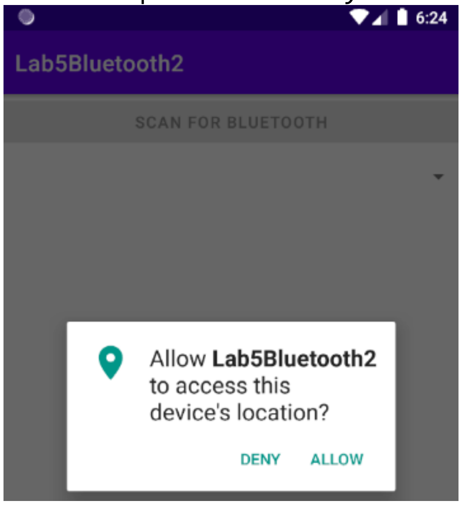
```
// Register the permissions callback, which handles the user's response to the
// system permissions dialog. Save the return value, an instance of
// ActivityResultLauncher, as an instance variable.
private final ActivityResultLauncher<String> requestPermissionLauncher =
        registerForActivityResult(new ActivityResultContracts.RequestPermission(), isGranted -> {
            if (isGranted) {
                scanBtn.setEnabled(true);
            } else {
                // Send a toast message to the user telling them they will not be able to use the app without allowing fine location permission.
                // There are multiple ways to do this, view the textbook for more information https://ebookcentral-proquest-com.ez.library.latrobe.
                Toast.makeText( context: this,  text: "Essential permission not granted please restart the app to try again",Toast.LENGTH_SHORT).show();
            }
        });
```

g) But how do we launch the launcher? Well we want it to run ONLY if permission have not already been granted so we can make an if statement in the onCreate() as we definitely want it to run at least once. Here we are using the hasPermission() function we created earlier to check permission has been granted, if not we make the button inactive as it wouldn't work anyway and launch the requestpermissionlauncher passing in the permission we are asking for.



```
//Get Permission to use bluetooth
if (!hasPermission()) {
    scanBtn.setEnabled(false); //Deactivate button if permission not granted
    //ask permissions
    Log.d(TAG,  msg: "Asking Real time permission");
    requestPermissionLauncher.launch(Manifest.permission.ACCESS_FINE_LOCATION);
}
```

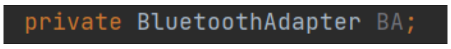
Now run your app, you should be asked permission on your first run like below;



Now we have finally finished the setup we can start working on the scan functionality. Yay!

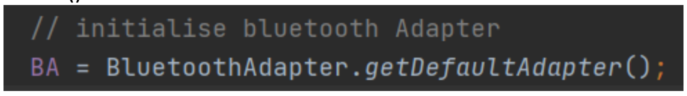
STEP 2 – SCAN FOR BLUETOOTH DEVICES

a) As we are dealing with Bluetooth add a Bluetooth member variable



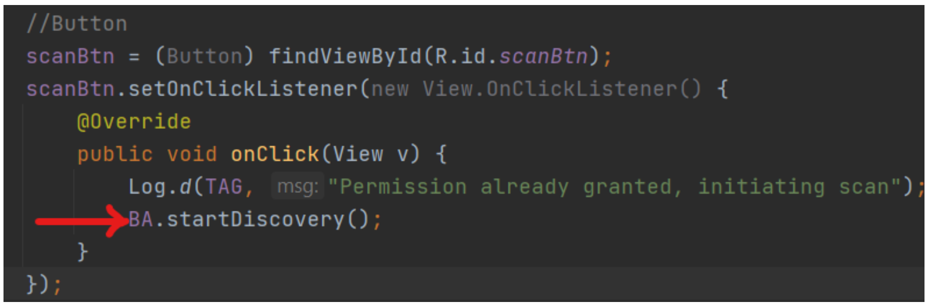
```java
private BluetoothAdapter BA;
```

b) And initalise it inside the onCreate()



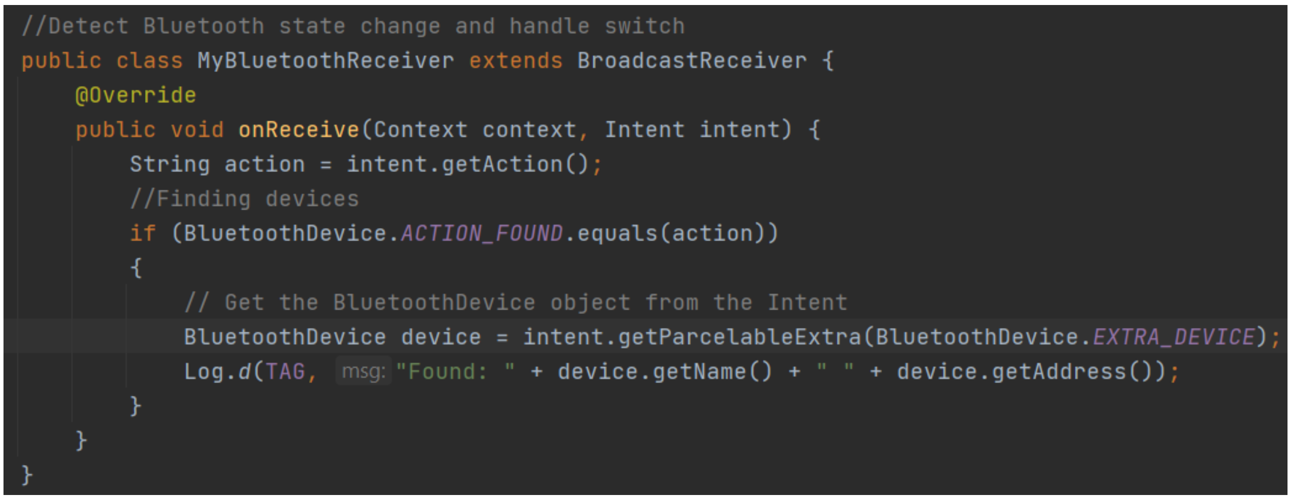
```java
// initialise bluetooth Adapter
BA = BluetoothAdapter.getDefaultAdapter();
```

c) Now you can put the call to BA.startDiscovery() inside your button onClick() to start looking for bluetooth discoverable devices.
NOTE: Please view the startDiscovery() documentation here, as we are targeting Android 11 (API 30 Build code R) we have the correct permissions, however for builds after Android 11 there we be some permission changes. Google is great at this, I cannot encourage you enough to read the docs for functions as you code as things change all the time!



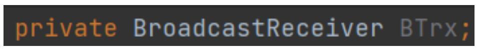
```java
//Button
scanBtn = (Button) findViewById(R.id.scanBtn);
scanBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Log.d(TAG, msg: "Permission already granted, initiating scan");
        BA.startDiscovery();
    }
});
```

d) When as device has been located an "android.bluetooth.device.action.FOUND" broadcast will be generated, so we will need a broadcast receiver, and as we want to interact easily with the activity we will make it a subclass inside the MainActivity like we did last week.



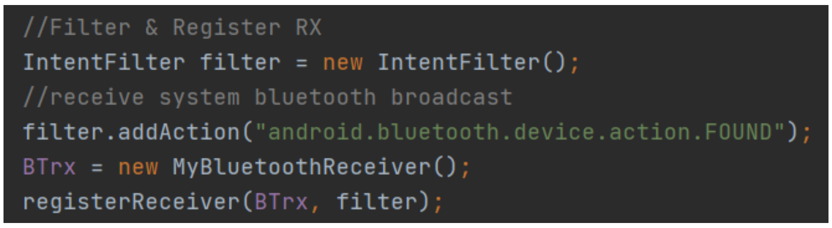
```java
//Detect Bluetooth state change and handle switch
public class MyBluetoothReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        //Finding devices
        if (BluetoothDevice.ACTION_FOUND.equals(action))
        {
            // Get the BluetoothDevice object from the Intent
            BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
            Log.d(TAG, msg: "Found: " + device.getName() + " " + device.getAddress());
        }
    }
}
```

e) And of course as we have a Broadcast Receiver we need an……….? Intent filter yes and to register our Broadcast receiver. First we must declare the Broadcast receiver with the rest of our member variables



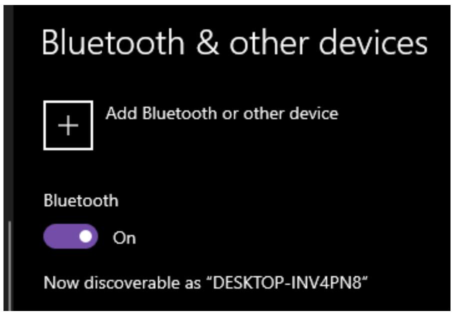
```java
private BroadcastReceiver BTrx;
```

f) And add the filter and registerReceiver method into the onCreate() in the MainActivity.



```
//Filter & Register RX
IntentFilter filter = new IntentFilter();
//receive system bluetooth broadcast
filter.addAction("android.bluetooth.device.action.FOUND");
BTrx = new MyBluetoothReceiver();
registerReceiver(BTrx, filter);
```

Now you should be able to detect discoverable Bluetooth devices BUT they will need to be set as discoverable. The devices will appear in your logcat. Not all devices will have a name, they may only return the address



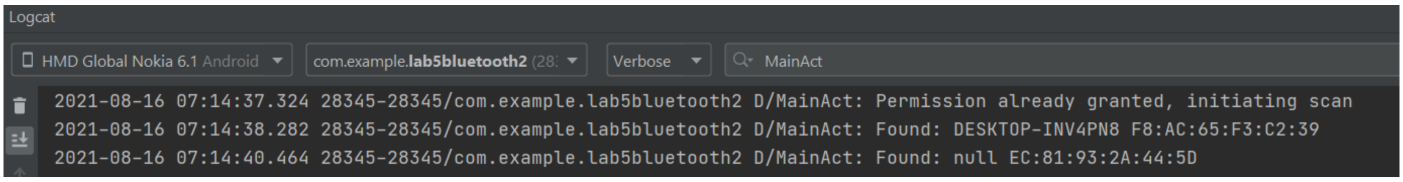
Bluetooth & other devices

Add Bluetooth or other device

Bluetooth

On

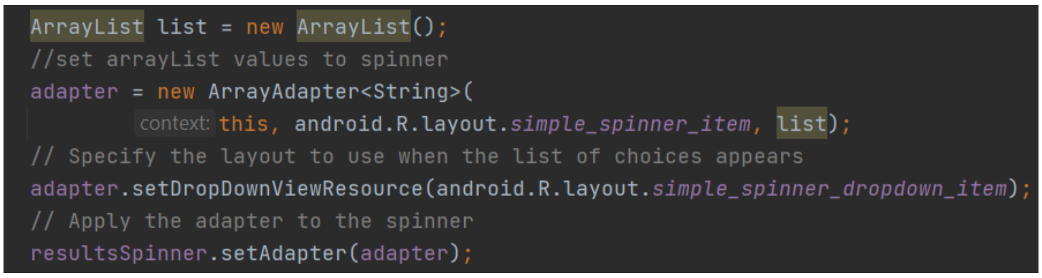
Now discoverable as "DESKTOP-INV4PN8"



```
2021-08-16 07:14:37.324 28345-28345/com.example.lab5bluetooth2 D/MainAct: Permission already granted, initiating scan
2021-08-16 07:14:38.282 28345-28345/com.example.lab5bluetooth2 D/MainAct: Found: DESKTOP-INV4PN8 F8:AC:65:F3:C2:39
2021-08-16 07:14:40.464 28345-28345/com.example.lab5bluetooth2 D/MainAct: Found: null EC:81:93:2A:44:5D
```

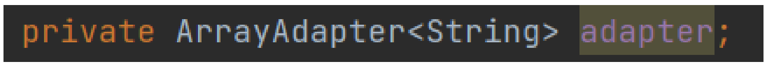
STEP 3 – SET UP SPINNER & ARRAY ADAPTER

a) Now that we are getting results it would be nice to display them somewhere.. a spinner perhaps.. Thus far we have linked the spinner object to the View in the onCreate() but some additional preparation is required. Add the following code into the onCreate() under the linked spinner object. Here we are making an ArrayList to store the information we have received. Next we create an arrayAdapter where we specify the xml of the spinner and the list we created earlier. Then we set the adapter to the results spinner.



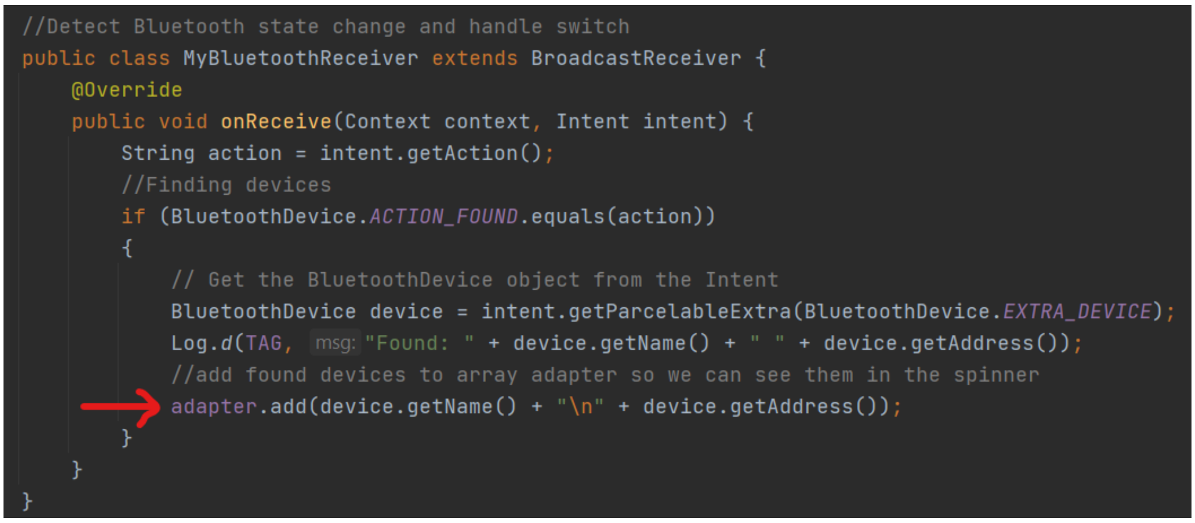
```
ArrayList list = new ArrayList();
//set arrayList values to spinner
adapter = new ArrayAdapter<String>(
        context: this, android.R.layout.simple_spinner_item, list);
// Specify the layout to use when the list of choices appears
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
// Apply the adapter to the spinner
resultsSpinner.setAdapter(adapter);
```

b) We will also need to add the adapter as a member variable!



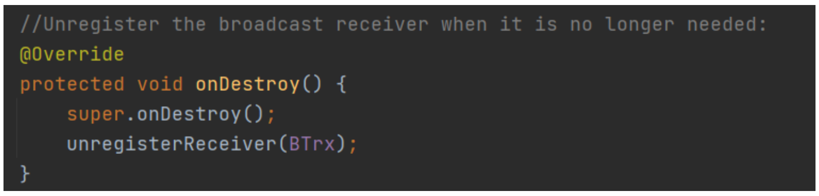
```
private ArrayAdapter<String> adapter;
```

c) Now in the Broadcast receiver we have to add the devices to the arraylist adapter as they are found, this will propagate the spinner.



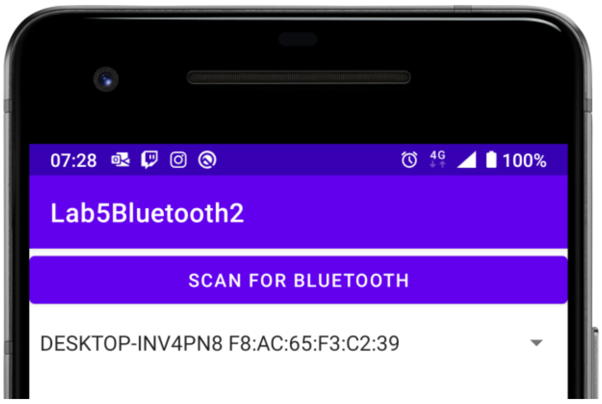
```java
//Detect Bluetooth state change and handle switch
public class MyBluetoothReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        //Finding devices
        if (BluetoothDevice.ACTION_FOUND.equals(action))
        {
            // Get the BluetoothDevice object from the Intent
            BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
            Log.d(TAG,  msg: "Found: " + device.getName() + " " + device.getAddress());
            //add found devices to array adapter so we can see them in the spinner
            adapter.add(device.getName() + "\n" + device.getAddress());
        }
    }
}
```

d) And lastly we must unregister the broadcast receiver in the onDestroy()



```java
//Unregister the broadcast receiver when it is no longer needed:
@Override
protected void onDestroy() {
    super.onDestroy();
    unregisterReceiver(BTrx);
}
```

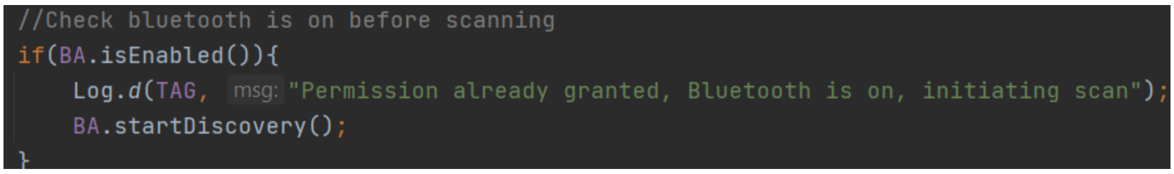
e) Now you should have a functioning app ready to scan all your devices 😉



CSE2MAD LAB 5 – CHECK FOR BLUETOOTH

Logically if an app is scanning for Bluetooth devices, it would be wise to have Bluetooth on so we can connect to them.

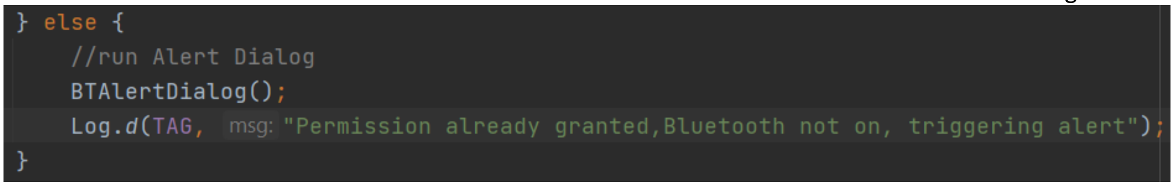
a) We should do this as soon as they press the button to scan so we should use a conditional statement in the onClick, as we don't want to activate the Bluetooth when the app is just opened as to not waste the battery. Though its really up to you 😊 In this instance we will check the state of the Bluetooth using the isEnabled method of the Bluetooth Adapter class, read more here https://developer.android.com/reference/android/bluetooth/BluetoothAdapter#isEnabled()



```
//Check bluetooth is on before scanning
if(BA.isEnabled()){
    Log.d(TAG, msg: "Permission already granted, Bluetooth is on, initiating scan");
    BA.startDiscovery();
}
```

b) Hmm so what if they don't have it enabled? It's a bit rude to just enable things without asking so we can make an Alert Dialog. Dialogs as very useful tools which are quite simple to build. Please read more about Dialogs here https://developer.android.com/guide/topics/ui/dialogs

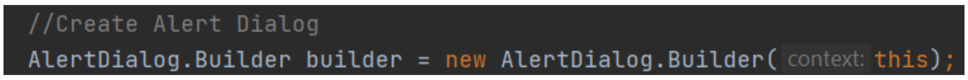
Now we should add and else the above is statement that will call the custom method to create the dialog.



```
} else {
    //run Alert Dialog
    BTAlertDialog();
    Log.d(TAG, msg: "Permission already granted,Bluetooth not on, triggering alert");
}
```

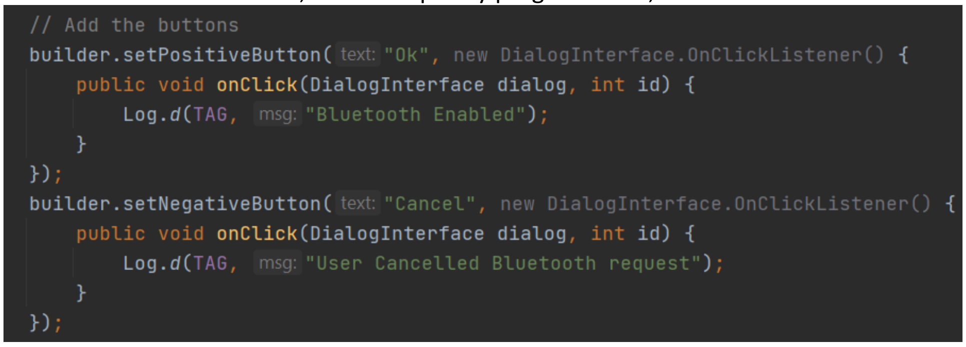
c) See an error? Yes we need to create the BTAlertDialog function that we have called in the else statement. Please create it OUTSIDE the onCreateas it does not need to run as soon as the Activity is opened. There is no return required so it will be void.

d) First create an AlertDialog builder object, we will use this builder object to set the Dialog attributes to.



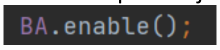
```
//Create Alert Dialog
AlertDialog.Builder builder = new AlertDialog.Builder( context: this);
```

e) Now we can add the buttons to the builder, these are purely programmatic, so we set the button text here too.



```
// Add the buttons
builder.setPositiveButton( text: "Ok", new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
        Log.d(TAG, msg: "Bluetooth Enabled");
    }
});
builder.setNegativeButton( text: "Cancel", new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
        Log.d(TAG, msg: "User Cancelled Bluetooth request");
    }
});
```

f) So at the moment the buttons just create logcat messages, in which onClick would you place the code below to enable Bluetooth? We are just calling enable() on the Bluetooth adapter object we already created.



```
BA.enable();
```

g) Now we only have button but the user needs more information, we van use the builder object to add more details to the dialog

```
builder.setMessage("Please turn on Bluetooth to use scanner")
        .setTitle("Alert");
```

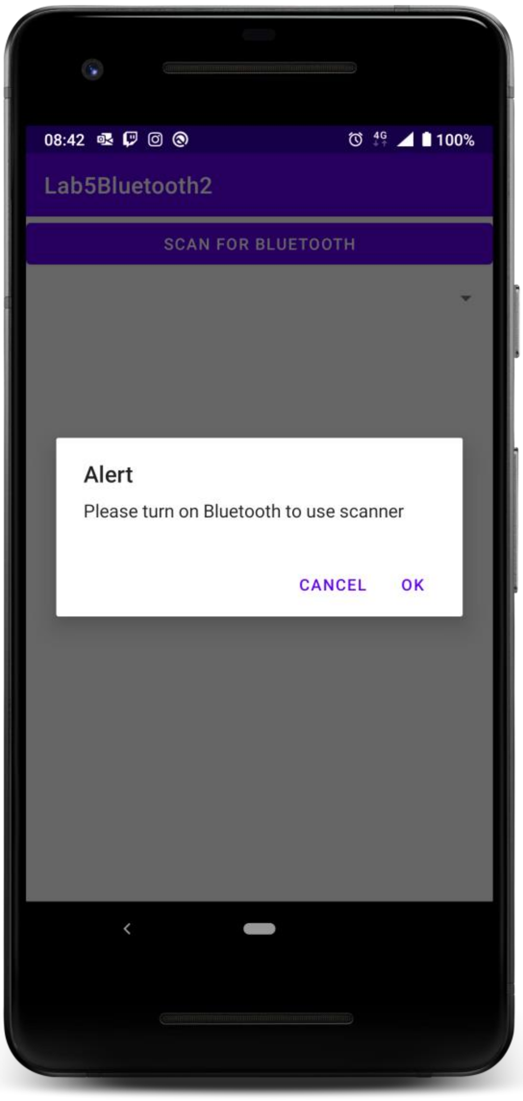
h) We then have to create the dialog from the builder object

```
// Create the AlertDialog
AlertDialog dialog = builder.create();
```

i) And now we have completed the dialog we can show it, completing out BTAlertDialog() function

```
//Start the dialog
dialog.show();
```

j) Now you should have an Alert Dialog that turns on the Bluetooth if the user presses OK



Alert

Please turn on Bluetooth to use scanner

CANCEL OK