Stacks on Stacks

Group Members:

Ian Matthews 15302424

Akua Afrane-Okese 15019773
Ntiko Mathaba 14012503
Linda Zwane 14199468

Testing Policy Document

How the tests are being carried out

For testing, the focus is on making sure functions run with the different possible cases. The expected result is given before the application is installed and run the application on an Android mobile phone. Tests are performed immediately after coding the classes to mitigate errors in advance together with each time an update is made.

What is being Tested ?

The following scenarios are tested on:

1. Device Scanning
2. Bluetooth Connectivity and Bluetooth Initialization
3. Device Control
4. Gatt Attributes

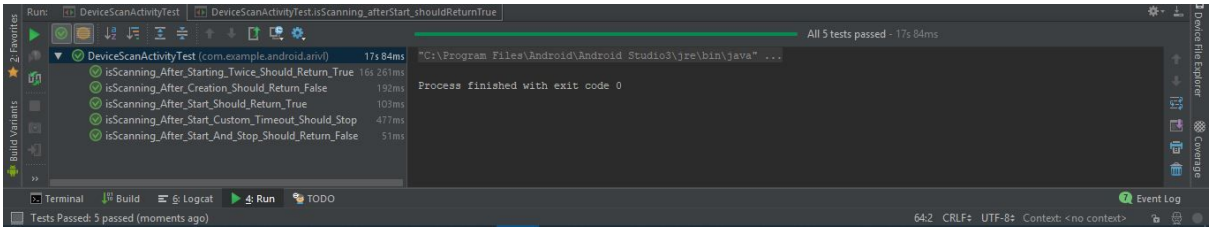
1. Device Scanning

- Testing if it is scanning only after creation (Expected Result: False)
- Testing if scanning after creation and instructed to scan for BLE device(Expected Result: True)
- Testing if device continues scanning after instructed to twice (Expected Result: True)
- Testing if device is scanning after instructed to start and then stop scanning(Expected Result: False)
- Testing to verify if device is scanning a custom timeout period(Expected Result:False)

```
import ...

@RunWith(RobolectricTestRunner.class)
@Config(constants = BuildConfig.class,sdk=18,manifest = "src/main/AndroidManifest.xml", packageName = "com.example.android.arivl")
public class DeviceScanActivityTest {

    @Test public void isScanning_After_Creation_Should_Return_False() {
        BluetoothAdapter adapter = Mockito.mock(BluetoothAdapter.class);
        LeScanCallback callback = Mockito.mock(LeScanCallback.class);
        com.example.android.arivl.DeviceScanActivity bleDevicesScanner = new com.example.android.arivl.DeviceScanActivity(adapter,callback);
        Assert.assertFalse(bleDevicesScanner.isScanning());
    }

    @Test public void isScanning_After_Start_Should_Return_True() {
        BluetoothAdapter adapter = Mockito.mock(BluetoothAdapter.class);
        LeScanCallback callback = Mockito.mock(LeScanCallback.class);
        Handler handler = Mockito.mock(Handler.class);
        com.example.android.arivl.DeviceScanActivity bleDevicesScanner = new com.example.android.arivl.DeviceScanActivity(adapter,callback);
        bleDevicesScanner.start(handler);
        Assert.assertTrue(bleDevicesScanner.isScanning());
    }
```

```
    }

    @Test public void isScanning_After_Starting_Twice_Should_Return_True() {
        BluetoothAdapter adapter = Mockito.mock(BluetoothAdapter.class);
        LeScanCallback callback = Mockito.mock(LeScanCallback.class);
        Handler handler = Mockito.mock(Handler.class);
        com.example.android.arivl.DeviceScanActivity bleDevicesScanner = new com.example.android.arivl.DeviceScanActivity(adapter,callback);
        bleDevicesScanner.start(handler);
        bleDevicesScanner.start(handler);
        Assert.assertTrue(bleDevicesScanner.isScanning());
    }

    @Test public void isScanning_After_Start_And_Stop_Should_Return_False() {
        BluetoothAdapter adapter = Mockito.mock(BluetoothAdapter.class);
        LeScanCallback callback = Mockito.mock(LeScanCallback.class);
        Handler handler = Mockito.mock(Handler.class);
        com.example.android.arivl.DeviceScanActivity bleDevicesScanner = new com.example.android.arivl.DeviceScanActivity(adapter,callback);
        bleDevicesScanner.start(handler);
        bleDevicesScanner.stop(handler);
        Assert.assertFalse(bleDevicesScanner.isScanning());
    }
```

```
    @Test public void isScanning_After_Start_Custom_Timeout_Should_Stop() {
        BluetoothAdapter adapter = Mockito.mock(BluetoothAdapter.class);
        LeScanCallback callback = Mockito.mock(BluetoothAdapter.LeScanCallback.class);
        Handler handler = Mockito.mock(Handler.class);
        com.example.android.arivl.DeviceScanActivity bleDevicesScanner = new com.example.android.arivl.DeviceScanActivity(adapter,callback);
        bleDevicesScanner.setScanPeriod(1);
        bleDevicesScanner.start(handler);
        Assert.assertTrue(bleDevicesScanner.isScanning());
        //timeout for Mockito in milliseconds
        Mockito.verify(adapter, Mockito.timeout( millis: 2000).atLeastOnce()).stopLeScan(bleDevicesScanner.get(adapter,handler));
    }
}
```



2. Device Connectivity

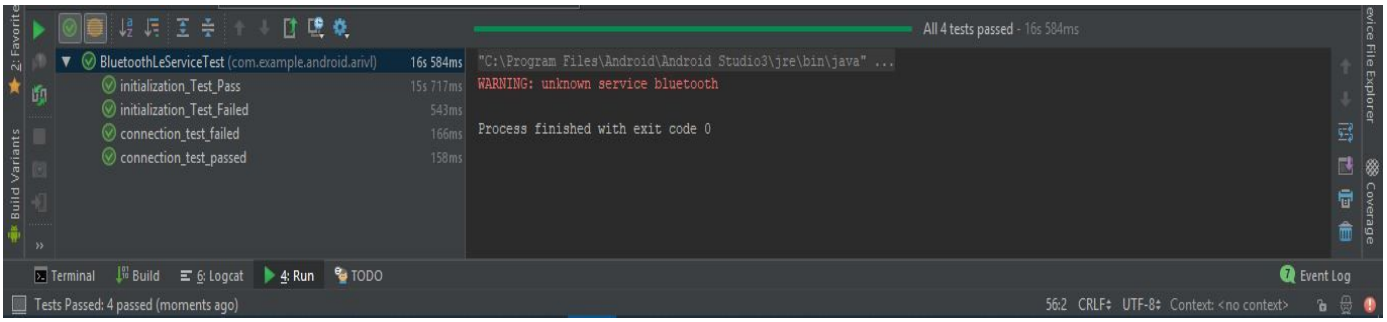
- testing if connection is established (Expected Result : True)
- testing if connection is not established (Expected Result: False)
- testing if bluetooth initialized (Expected Result: True)
- testing if bluetooth not initialized (Expected Result: False)

```
package com.example.android.arivl;

import ...

@RunWith(RobolectricTestRunner.class)
@Config(constants = BuildConfig.class,sdk=18,manifest = "src/main/AndroidManifest.xml", packageName = "com.example.android.arivl")
public class BluetoothLeServiceTest{
    @Test
     public void connection_test_passed(){
        BluetoothAdapter adapter = Mockito.mock(BluetoothAdapter.class);
        BluetoothGatt gatt = Mockito.mock(BluetoothGatt.class);
        when(gatt.connect()).thenReturn(true);
        com.example.android.arivl.BluetoothLeService leService = new com.example.android.arivl.BluetoothLeService(adapter,gatt);
        Assert.assertTrue(leService.connect( address: "Arivl"));
    }
```

```
    @Test
    public void connection_test_failed(){
        BluetoothAdapter adapter = Mockito.mock(BluetoothAdapter.class);
        BluetoothGatt gatt = Mockito.mock(BluetoothGatt.class);
        when(gatt.connect()).thenReturn(true);
        com.example.android.arivl.BluetoothLeService leService = new com.example.android.arivl.BluetoothLeService();
        Assert.assertFalse(leService.connect( address: "Arivl"));
    }

    @Test
    public void initialization_Test_Pass(){
        Context context = spy(RuntimeEnvironment.application);
        BluetoothManager manager = Mockito.mock(BluetoothManager.class);
        when(manager.getAdapter()).thenReturn(Mockito.mock(BluetoothAdapter.class));
        com.example.android.arivl.BluetoothLeService leService = new com.example.android.arivl.BluetoothLeService(manager);
        Assert.assertTrue(leService.initialize(context));
    }
```

```
    @Test
    public void initialization_Test_Failed(){
        Context context = spy(RuntimeEnvironment.application);
        BluetoothManager service = Mockito.mock(BluetoothManager.class);
        when(context.getSystemService(Context.BLUETOOTH_SERVICE)).thenReturn(service);
        com.example.android.arivl.BluetoothLeService leService = new com.example.android.arivl.BluetoothLeService();
        Assert.assertFalse(leService.initialize(context));
    }
}
```

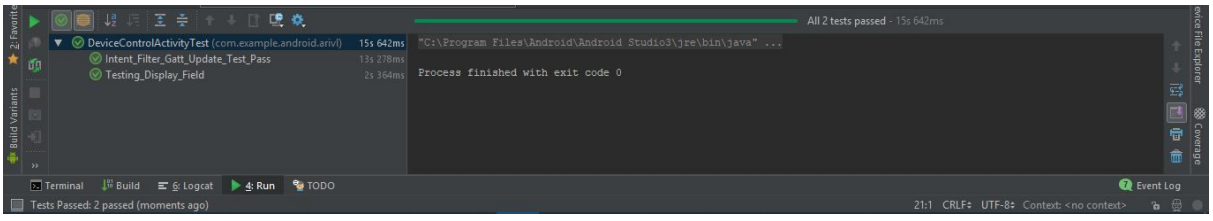
(Above are images of the code for the test cases)



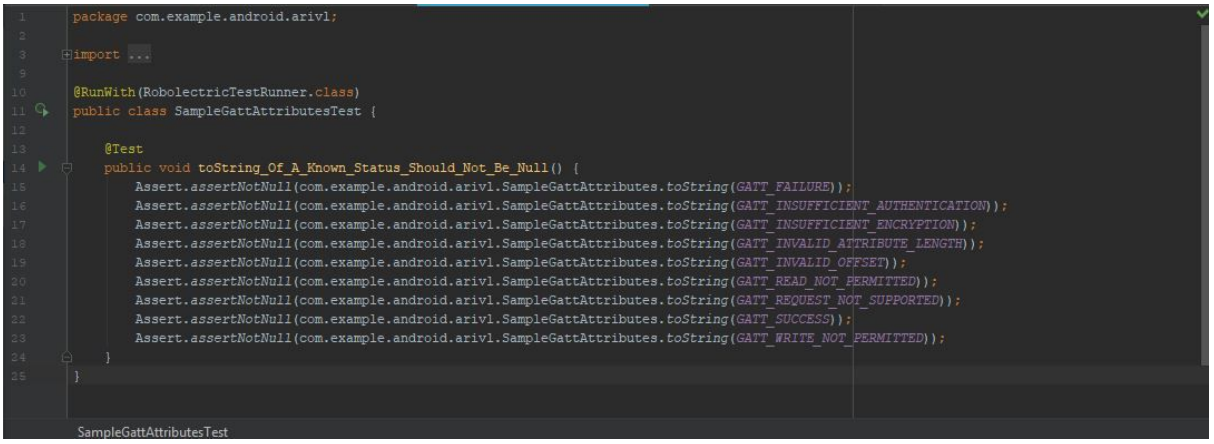
(Above is a screenshot of all 4 Test Cases Passing for Connectivity)

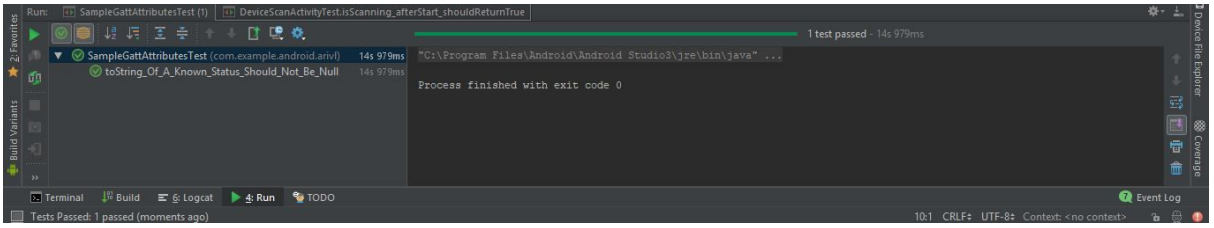
3. Device Control
   - To ensure Intent Filter updates (Expected Result : NotNull)
   - To ensure the display field is updated (Expected Result : NotNull)

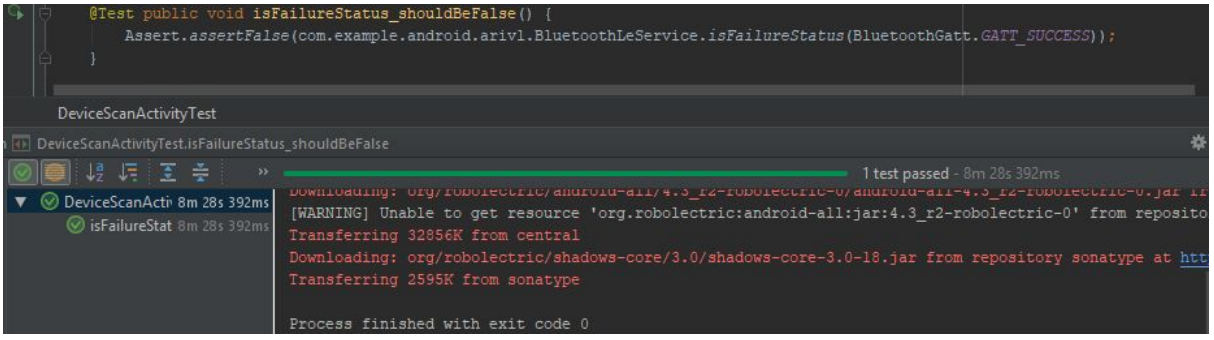
```
package com.example.android.arivl;

import ...
@RunWith(RobolectricTestRunner.class)
@Config(constants = BuildConfig.class,sdk=18,manifest = "src/main/AndroidManifest.xml", packageName = "com.example.android.arivl")
public class DeviceControlActivityTest {


    @Test
    public void Intent_Filter_Gatt_Update_Test_Pass(){
        // com.example.android.arivl.DeviceControlActivity deviceControlActivity = new com.example.android.arivl.DeviceControlActivity();
        Assert.assertNotNull(com.example.android.arivl.DeviceControlActivity.gatt());
    }

    @Test
    public void Testing_Display_Field(){
        com.example.android.arivl.DeviceControlActivity deviceControlActivity = new com.example.android.arivl.DeviceControlActivity();
        com.example.android.arivl.DeviceControlActivity deviceControlActivity1 = Mockito.mock(com.example.android.arivl.DeviceControlActivity.cl
        deviceControlActivity1.display( data: "ArivlApp");
    }
}
```



4. Gatt Attributes
   - To ensure read/write permissions (Expected Result : NotNull)
   - To ensure sufficient encryption and authentication (Expected Result : NotNull)
   - To ensure request permissions (Expected Result : NotNull)

- 

- 

- 

Testing Tools

- JUnit Testing Framework (4.12)
  The statement to include its functionality is: testImplementation 'junit:junit:4.12' (see in Figure 1).To use JUnit, its statement needs to be included in the Android Studio project gradle, in the section called dependency.

  JUnit allows for programmers to work on applications and their testing from one platform, which allows access to package private elements within the application. It provides annotations to easily distinguish between the application and testing

- Mockito (1.10.9)
  The statement to include its functionality is: testImplementation 'org.mockito:mockito-core:1.10.19' (see in Figure 1). To include its functionality, its statement needs to be included in the Android Studio project gradle, in the dependency section.

  Mockito is used to mock interfaces so that dummy functionalities (or functionality) can be added to a mock interface that can be used in unit testing so that classes can be tested on their own, without testing dependencies to verify that the code being tested

works without dependencies. It also caters for resources that are not actually available.

- Espresso (3.0.2)
  Espresso also needs to be stated in the dependency section. Its statement being: androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2' (see in Figure 1).

  Espresso is used to create automated user interface tests. It is based on JUnit, thus easy to include.

- Robolectric (3.0)
  Robolectric also needs to be stated in the dependency section. Here is its statement: androidTestImplementation 'org.robolectric:robolectric:3.0'.

  Not all areas need to be mocked (use mocking frameworks like Mockito). Robolectric lets you run your tests on your workstation, or on your Continuous Integration environment in a regular JVM, without an emulator.

```
dependencies {
    implementation "com.android.support:support-v4:27.0.2"
    implementation "com.android.support:support-v13:27.0.2"
    implementation "com.android.support:cardview-v7:27.0.2"
    implementation "com.android.support:appcompat-v7:27.0.2"
    implementation 'com.android.support.constraint:constraint-layout:1.1.0'
    implementation 'com.android.volley:volley:1.0.0'
    androidTestImplementation 'com.android.support.test:runner:1.0.2'
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'

    testImplementation 'org.mockito:mockito-core:1.10.19'
    testImplementation 'junit:junit:4.12'
    testImplementation 'org.robolectric:robolectric:3.0'
    androidTestImplementation 'org.robolectric:robolectric:3.0'
}
```

Figure 1: StacksonStacks Project Gradle in Android Studio

Test cases:
All the tests can be found on Github:
https://github.com/devawa/StacksOnStacks/tree/master/Android_Dev/Tests

References

Android Developers. [n.d]. Espresso. [Online]. Available: https://developer.android.com/training/testing/espresso/ [Accessed 18 July 2018].
Robolectric. [n.d]. Robolectric Test-Drive your Android Code. [Online]. Available: http://robolectric.org/ [Accessed 19 July 2018].

Tutorialspoint. [n.d]. Mockito Tutorial. [Online]. Available: https://www.tutorialspoint.com/mockito/ [Accessed 19 July 2018].