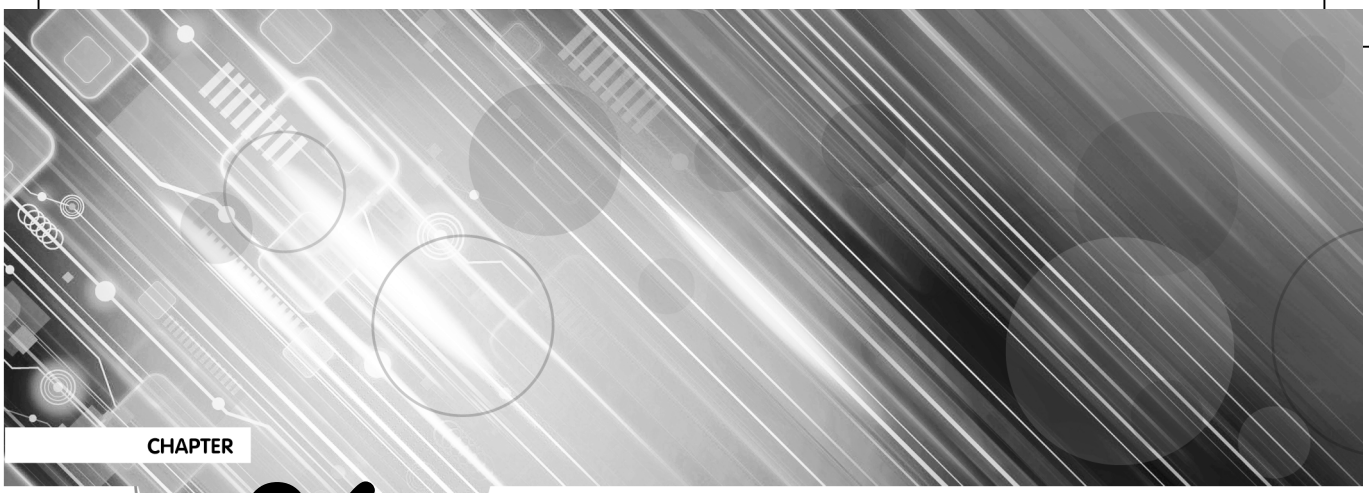


CHAPTER

06 โทรศัพท์ และ SMS

เนื้อหาในบทนี้

- การโทรออกด้วยโค้ด
- การตรวจสอบสถานะการโทร
- การตรวจสอบสถานะการโทรแบบหลังฉาก
- การปิดกั้นการโทรออกไปยังหมายเลขหนึ่งๆ
- การรับสายอัตโนมัติ
- การหาหมายเลขโทรศัพท์, รหัสซิมการ์ด และหมายเลข IMEI ของเครื่อง
- การส่ง SMS โดยใช้แอพที่มีอยู่ในเครื่อง
- การส่ง SMS จากแอพของเราเอง
- การตรวจสอบสถานะของ SMS ที่ส่งออกไป
- การตรวจสอบ SMS ที่ส่งออกไปจากเครื่อง
- การรับ SMS

การโทรออกด้วยโค้ด

เราสามารถเขียนโค้ดเพื่อสั่งโทรออกไปยังหมายเลขที่ต้องการได้ ตัวอย่างการใช้งานคือ สมมติเราสร้างแอพที่แสดงข้อมูลเกี่ยวกับสถานที่ต่างๆ เช่น ร้านอาหาร โรงพยาบาล โดยหนึ่งในข้อมูลเหล่านั้นคือหมายเลขโทรศัพท์ เราอาจเตรียมปุ่มไว้ให้ผู้ใช้โทรไปยังหมายเลขนั้นได้ทันที โดยผู้ใช้ไม่ต้องเรียกหน้าจอโทรศัพท์และกดหมายเลขเอง

ตัวอย่าง

ตัวอย่างนี้จะมี `EditText` ไว้ให้กรอกหมายเลขโทรศัพท์ จากนั้นเมื่อคลิกปุ่มก็จะโทรออกไปยังหมายเลขนั้น

1 กำหนด Layout ของหน้าจอ

โปรเจ็ค MakePhoneCall, ไฟล์ activity__main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp" >

    <EditText
        android:id="@+id/phone_number"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="phone" >

        <requestFocus />
    </EditText>

    <Button
        android:id="@+id/make_call_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="8dp"
        android:text="Make Call" />

</LinearLayout>
```

2 เพิ่มโค้ดในเมธอด onCreate ของแอคทิวิตี

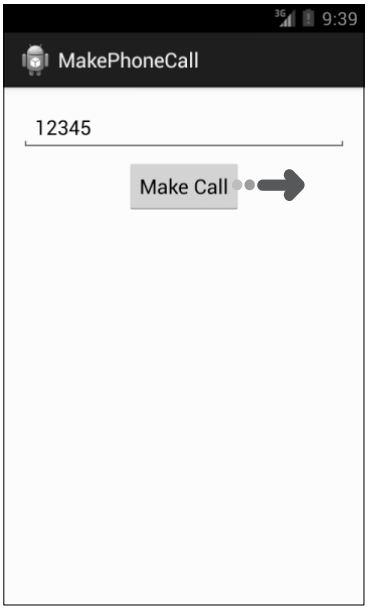
โปรเจ็ค MakePhoneCall, ไฟล์ MainActivity.java

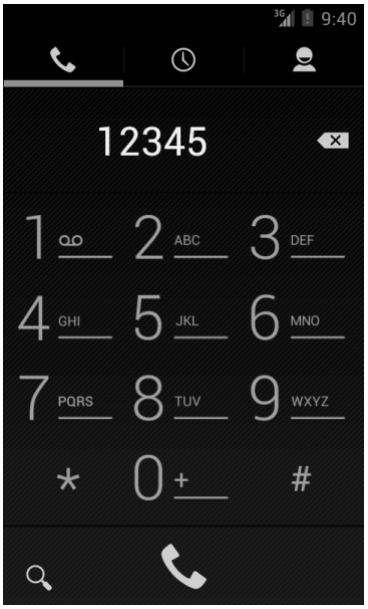
```
Button btnMakeCall = (Button) findViewById(R.id.make_call_button);
btnMakeCall.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View v) {
        TextView txtPhoneNumber = (TextView)
                                  findViewById(R.id.phone_number);
        String uri = "tel:" + txtPhoneNumber.getText();
        Intent intent = new Intent(Intent.ACTION_DIAL, Uri.parse(uri));
        startActivity(intent);
    }
});
```

ผลการรัน

เมื่อกรอกหมายเลขแล้วคลิกปุ่ม Make Call จะปรากฏหน้าจอโทรศัพท์ขึ้นมา ดังรูป





คำอธิบาย

การโทรออก ให้สร้างอินเทนต์โดยกำหนดแอคชั่นเป็น ACTION_DIAL และกำหนดข้อมูลของอินเทนต์เป็นออบเจ็ค Uri ที่ parse มาจากค่าสตริงในรูปแบบ "tel:*หมายเลขโทรศัพท์*" เช่น

```
Intent intent = new Intent();              // สร้างอินเทนต์
intent.setAction(Intent.ACTION_DIAL);      // กำหนดแอคชั่นของอินเทนต์
intent.setData(Uri.parse("tel:12345"));    // กำหนดข้อมูลของอินเทนต์
```

หรืออาจกำหนดแอคชั่นและข้อมูลของอินเทนต์ตั้งแต่ตอนสร้างอินเทนต์เลย ซึ่งจะทำให้โค้ด 3 บรรทัดข้างต้นเหลือเพียงบรรทัดเดียว ดังนี้

```
Intent intent = new Intent(Intent.ACTION_DIAL, Uri.parse("tel:12345"));
```

หลังจากสร้างอินเทนต์แล้ว ให้เรียกเมธอด startActivity โดยระบุอินเทนต์นั้นเป็นพารามิเตอร์

```
startActivity(intent);
```

แอนดรอยด์ก็จะรันแอคทิวิตีที่สามารถจัดการ ACTION_DIAL ได้ขึ้นมา (ปกติคือหน้าจอโทรศัพท์หรือแอพ Phone นั่นเอง)

การสั่งโทรออกทันที

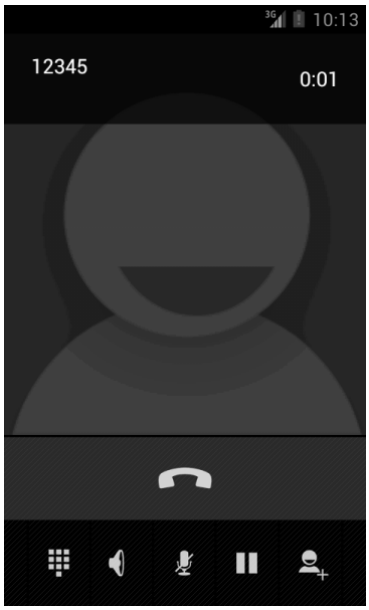
ตัวอย่างเมื่อครู่นี้จะเรียกหน้าจอโทรศัพท์ขึ้นมาแต่ไม่โทรออกให้ทันที ผู้ใช้ต้องกดปุ่มโทรออกเอง แต่ถ้าคุณต้องการโทรออกทันที ให้เปลี่ยนแอคชั่นของอินเทนต์เป็น ACTION_CALL ดังตัวอย่าง

```
Intent intent = new Intent(Intent.ACTION_CALL, Uri.parse("tel:12345"));
```

เมื่อใช้ ACTION_CALL คุณจะต้องขอสิทธิ์การโทรออกจากแอนดรอยด์ด้วย (สำหรับ ACTION_DIAL ไม่ต้องขอสิทธิ์ใดๆเป็นพิเศษ) ดังนั้นให้เพิ่มบรรทัดนี้ในไฟล์ AndroidManifest.xml โดยใส่ไว้ก่อนแท็ก <application>

```
<uses-permission android:name="android.permission.CALL_PHONE" />
```

ทดลองรันแอพอีกครั้ง จากนั้นกรอกหมายเลข แล้วคลิกปุ่ม Make Call หน้าจอโทรศัพท์จะปรากฏขึ้นมาและโทรออกทันที ดังรูป



อย่างไรก็ตาม ในเอกสารของแอนดรอยด์จะแนะนำให้ใช้ ACTION_DIAL มากกว่า เนื่องจาก ACTION_DIAL สามารถโทรหมายเลขฉุกเฉินได้ ในขณะที่ ACTION_CALL โทรหมายเลขฉุกเฉินไม่ได้

ตรวจสอบว่าอุปกรณ์แอนดรอยด์นั้นโทรได้หรือไม่

สำหรับอุปกรณ์แอนดรอยด์ที่ไม่มีความสามารถด้านโทรศัพท์ เช่น แท็บเล็ตบางรุ่น แน่นอนว่าเมื่อรันตัวอย่างข้างต้นจะไม่สามารถโทรออกได้ ผู้เขียนทดสอบกับแท็บเล็ตตัวหนึ่ง (ซึ่งโทรไม่ได้) ก็พบว่าเมื่อคลิกปุ่ม Make Call จะปรากฏหน้าจอให้บันทึกหมายเลขลงในรายชื่อผู้ติดต่อแทน

เราสามารถตรวจสอบได้ว่าอุปกรณ์แอนดรอยด์ที่แอพของเรารันอยู่นั้นมีความสามารถด้านโทรศัพท์หรือไม่ โดยเพิ่มโค้ดในเมธอด onClick ของปุ่ม Make Call ดังนี้

โปรเจ็ค MakePhoneCall, ไฟล์ MainActivity.java

```
@Override
public void onClick(View v) {
    TelephonyManager telephony = (TelephonyManager)
                                 getSystemService(Context.TELEPHONY_SERVICE);

    if (telephony.getPhoneType() == TelephonyManager.PHONE_TYPE_NONE) {
        String msg = "Sorry, this device can't make phone calls!";
        Toast.makeText(MainActivity.this, msg, Toast.LENGTH_SHORT).show();
        return;
    }
```

```
    TextView txtPhoneNumber = (TextView) findViewById(R.id.phone_number);
    String uri = "tel:" + txtPhoneNumber.getText();
    Intent intent = new Intent(Intent.ACTION_DIAL, Uri.parse(uri));
    startActivity(intent);
}
```

จากโค้ด เราเรียกเมธอด `getSystemService` เพื่อเข้าถึงบริการ Telephony (บริการด้านโทรศัพท์) ของระบบแอนดรอยด์ จากนั้นเรียกเมธอด `getPhoneType` เพื่อตรวจสอบว่าอุปกรณ์นั้นคือโทรศัพท์ประเภทใด ซึ่งถ้าเป็นประเภท `PHONE_TYPE_NONE` ก็แสดงว่าไม่ใช่โทรศัพท์ (ไม่มีความสามารถด้านโทรศัพท์) เราจะแสดง Toast แจ้งผู้ใช้ แล้ว `return` ออกจากเมธอด `onClick` ทันที

การตรวจสอบสถานะการโทร

โทรศัพท์แอนดรอยด์จะมีสถานะการโทรเป็นอย่างใดอย่างหนึ่งใน 3 สถานะต่อไปนี้เสมอ

- **สถานะ Idle** เมื่อไม่มีกิจกรรมใดๆเกี่ยวกับการโทรเกิดขึ้น กล่าวคือ ไม่มีการโทรออก ไม่มีสายเรียกเข้า ไม่มีสายที่อยู่ระหว่างการสนทนาหรือรอการสนทนาอยู่
- **สถานะ Ringing** เมื่อมีสายเรียกเข้า
- **สถานะ Offhook** เมื่อรับสายเรียกเข้า หรือกำลังโทรออก

แอนดรอยด์มีวิธีให้เราตรวจสอบสถานะการโทร เพื่อระบุการทำงานที่ต้องการเมื่อโทรศัพท์เข้าสู่สถานะต่างๆ ยกตัวอย่างเช่น เมื่อมีสายเรียกเข้าในขณะที่ผู้ใช้ขับรถอยู่ แอพของเราอาจส่ง SMS ไปยังหมายเลขที่เรียกเข้ามาโดยอัตโนมัติ เพื่อบอกว่าผู้ใช้ไม่สะดวกที่จะรับสาย เป็นต้น

ตัวอย่างและคำอธิบาย

ตัวอย่างนี้จะทำต่อจากตัวอย่างที่แล้ว โดยเพิ่มการตรวจสอบสถานะการโทรและจะแสดง Toast เมื่อสถานะการโทรเปลี่ยนจากสถานะหนึ่งไปเป็นอีกสถานะหนึ่ง (ในซอร์สโค้ดจะแยกออกมาเป็นโปรเจ็คใหม่ชื่อ CheckPhoneState)

1 สร้างคลาสใหม่ชื่อ `MyPhoneStateListener` แล้วพิมพ์โค้ดดังนี้

โปรเจ็ค CheckPhoneState, ไฟล์ MyPhoneStateListener.java

```
package com.example.checkphonestate;

import android.content.Context;
import android.telephony.PhoneStateListener;
import android.telephony.TelephonyManager;
import android.widget.Toast;
```

```
public class MyPhoneStateListener extends PhoneStateListener {
    Context context;

    public MyPhoneStateListener(Context context) {
        this.context = context;
    }

    @Override
    public void onCallStateChanged(int state, String incomingNumber) {
        super.onCallStateChanged(state, incomingNumber);

        switch (state) {
        case TelephonyManager.CALL_STATE_IDLE:
            showToast("Idle");
            break;
        case TelephonyManager.CALL_STATE_RINGING:
            showToast("Ringing from " + incomingNumber);
            break;
        case TelephonyManager.CALL_STATE_OFFHOOK:
            showToast("Offhook");
            break;
        }
    }

    private void showToast(String msg) {
        Toast.makeText(context, msg, Toast.LENGTH_SHORT).show();
    }
}
```

เราสร้างคลาส `MyPhoneStateListener` นี้โดยสืบทอดจากคลาส `PhoneStateListener` ของแอนดรอยด์ แล้ว Override เมธอด `onCallStateChanged` เพื่อระบุการทำงานเมื่อสถานะการโทรเปลี่ยนไป โดยพารามิเตอร์ `state` จะบอกสถานะการโทรปัจจุบัน (สถานะที่เพิ่งเปลี่ยนมาจากสถานะก่อนหน้า) และพารามิเตอร์ `incomingNumber` จะบอกหมายเลขเรียกเข้า กรณีที่สถานะปัจจุบันคือ Ringing (มีสายเรียกเข้า)

ในที่นี้จะแสดง Toast เป็นข้อความที่บอกถึงสถานะนั้นๆ และสำหรับสถานะ Ringing จะแสดงหมายเลขเรียกเข้าด้วย

2 ที่แอคทิวิตี ให้เพิ่มการประกาศตัวแปรระดับคลาส

```
TelephonyManager telephony;
MyPhoneStateListener listener;
```

3 เพิ่มโค้ดในเมธอด onCreate ถัดจากบรรทัด setContentView (ก่อนกำหนด Listener ให้กับปุ่ม Make Call)

โปรเจ็ค CheckPhoneState, ไฟล์ MainActivity.java

```
// เข้าถึงบริการ Telephony ของระบบแอนดรอยด์
telephony = (TelephonyManager) getSystemService(Context.TELEPHONY_SERVICE);
// สร้างออบเจ็คของคลาส MyPhoneStateListener
listener = new MyPhoneStateListener(this);
// นำออบเจ็คนั้นมาลงทะเบียนกับบริการ Telephony
telephony.listen(listener, PhoneStateListener.LISTEN_CALL_STATE);
```

คือการสร้างออบเจ็คของคลาส MyPhoneStateListener แล้วนำมาลงทะเบียนกับบริการ Telephony ของแอนดรอยด์ โดยระบุว่าให้แจ้งกลับมาเมื่อสถานะการโทรเปลี่ยนไป (LISTEN_CALL_STATE)

TIP

นอกจากสถานะการโทรแล้ว บริการ Telephony ของแอนดรอยด์ยังอนุญาตให้เราตรวจสอบการเปลี่ยนแปลงสถานะหรือข้อมูลอื่นๆที่เกี่ยวกับโทรศัพท์ได้อีก โดยระบุพารามิเตอร์ตัวที่ 2 ของเมธอด listen เป็นค่าคงที่ต่างๆ ได้แก่ (ค่าคงที่ทั้งหมดนี้ถูกกำหนดไว้ในคลาส PhoneStateListener)

- LISTEN_CALL_FORWARDING_INDICATOR
- LISTEN_CELL_INFO
- LISTEN_CELL_LOCATION
- LISTEN_DATA_ACTIVITY
- LISTEN_DATA_CONNECTION_STATE
- LISTEN_MESSAGE_WAITING_INDICATOR
- LISTEN_SERVICE_STATE
- LISTEN_SIGNAL_STRENGTHS

สำหรับความหมายและรายละเอียดเกี่ยวกับค่าคงที่เหล่านี้ สามารถศึกษาได้จากเอกสารของแอนดรอยด์ที่ developer.android.com/reference/android/telephony/PhoneStateListener.html

4 เพิ่มเมธอด onDestroy ในแอคทิวิตี

```
@Override
protected void onDestroy() {
    super.onDestroy();
    telephony.listen(listener, PhoneStateListener.LISTEN_NONE);
}
```

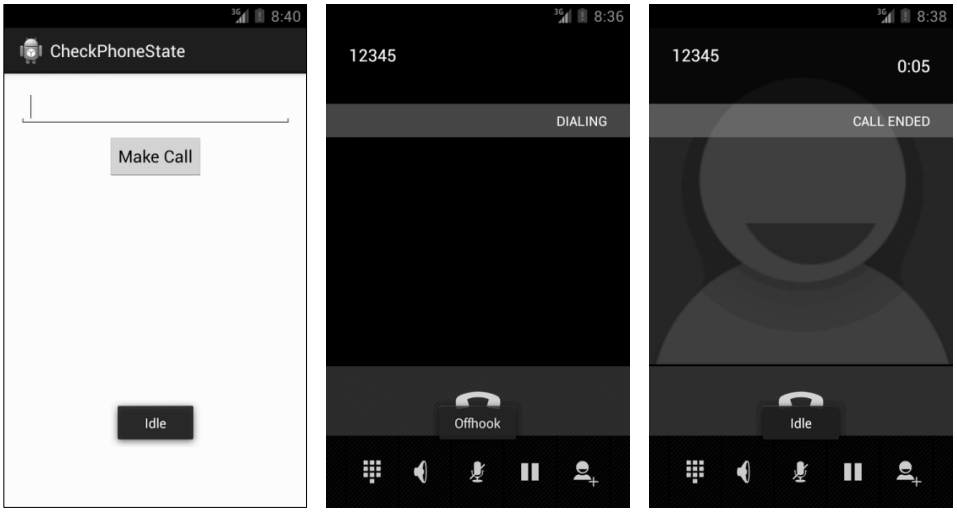
เมื่อแอคทิวิตีกำลังจะถูกปิด เราจะยกเลิกการลงทะเบียนออบเจ็ค MyPhoneStateListener เพื่อหยุดการตรวจสอบสถานะการโทร

5 เพิ่มบรรทัดต่อไปนี้ในไฟล์ AndroidManifest.xml โดยใส่ไว้ก่อนแท็ก `<application>` เพื่อขอสิทธิ์ในการอ่านสถานะของโทรศัพท์จากแอนดรอยด์

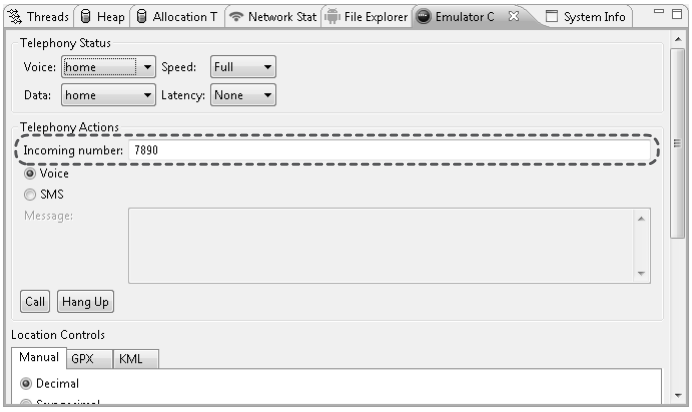
```
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

ผลการรัน

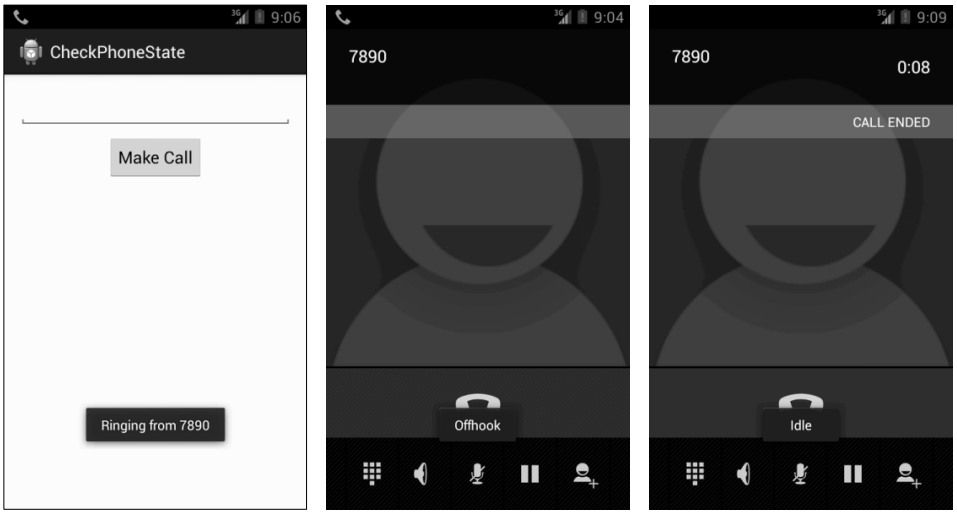
เมื่อรันแอพจะแจ้งสถานะ Idle ทันที (รูปซ้าย) จากนั้นเมื่อกรอกหมายเลข, คลิกปุ่ม Make Call แล้วกดโทรออกจะแจ้งสถานะ Offhook (รูปกลาง) และเมื่อกดวางสายจะกลับมาสถานะ Idle (รูปขวา)



สำหรับการโทรเข้า เราสามารถจำลองสายเรียกเข้าได้โดยไปที่แท็บ Emulator Control ใน DDMS Perspective แล้วป้อนหมายเลขที่สมมติเป็นสายเรียกเข้าในช่อง Incoming Number แล้วจึงคลิกปุ่ม Call



ที่หน้าจออีมูเลเตอร์จะแจ้งสถานะ Ringing พร้อมทั้งหมายเลขเรียกเข้า (รูปซ้าย) เมื่อกดรับสายจะแจ้งสถานะ Offhook (รูปกลาง) และเมื่อวางสายจะแจ้งสถานะ Idle (รูปขวา)



การตรวจสอบสถานะการโทรแบบหลังฉาก

การตรวจสอบสถานะการโทรโดยใช้ `PhoneStateListener` ในตัวอย่างที่ผ่านมาจะทำได้ก็ต่อเมื่อผู้ใช้รันแอพของเราแล้วเท่านั้น ด้วยเหตุนี้แอนดรอยด์จึงมีวิธีให้ตรวจสอบสถานะการโทรได้แบบหลังฉาก (Background) โดยใช้ `BroadcastReceiver` ซึ่งจะทำให้เราตรวจสอบสถานะการโทรได้ตลอดเวลาตราบใดที่แอพของเราติดตั้งอยู่ในเครื่องของผู้ใช้

ตัวอย่างและคำอธิบาย

ทุกครั้งที่สถานะการโทรเปลี่ยนไป แอนดรอยด์จะส่ง (broadcast) อินเทนต์ที่มีแอคชั่นเป็น `android.intent.action.PHONE_STATE` ออกมา ดังนั้นเราจะสร้าง `BroadcastReceiver` ที่คอยดักจับอินเทนต์ดังกล่าว

1 สร้างคลาสใหม่ชื่อ `MyBroadcastReceiver` แล้วพิมพ์โค้ดดังนี้

โปรเจ็ค CheckPhoneStateBackground, ไฟล์ MyBroadcastReceiver.java

```
package com.example.checkphonestatebackground;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
```

```
import android.telephony.TelephonyManager;
import android.widget.Toast;

public class MyBroadcastReceiver extends BroadcastReceiver {

    Context context;

    @Override
    public void onReceive(Context context, Intent intent) {
        this.context = context;

        String state =
            intent.getStringExtra(TelephonyManager.EXTRA_STATE); ❶

        if (state.equals(TelephonyManager.EXTRA_STATE_IDLE)) {
            showToast("Idle");
        } else if (state.equals(TelephonyManager.EXTRA_STATE_RINGING)) {
            String incomingNumber = intent.getStringExtra(
                        TelephonyManager.EXTRA_INCOMING_NUMBER); ❷
            showToast("Ringing from " + incomingNumber);
        } else if (state.equals(TelephonyManager.EXTRA_STATE_OFFHOOK)) {
            showToast("Offhook");
        }
    }

    private void showToast(String msg) {
        Toast.makeText(context, msg, Toast.LENGTH_SHORT).show();
    }
}
```

แอนดรอยด์จะใส่สถานะการโทรและหมายเลขเรียกเข้า (กรณีสถานะเป็น Ringing) ลงในอินเทนต์ที่ broadcast ออกมา และอินเทนต์นั้นจะถูกผ่านมายังพารามิเตอร์ตัวที่ 2 ของเมธอด `onReceive` ข้างต้น

เราจะอ่านสถานะการโทรจากอินเทนต์ ❶ แล้วเปรียบเทียบกับค่าคงที่ต่างๆเพื่อดูว่าเป็นสถานะใด แล้วแสดง Toast ออกมา ซึ่งหากเป็นสถานะ Ringing ก็จะอ่านหมายเลขเรียกเข้าจากอินเทนต์ ❷ มาแสดงใน Toast ด้วย

2 เพิ่มบรรทัดต่อไปนี้ในไฟล์ AndroidManifest.xml โดยใส่ไว้ก่อนแท็ก `<application>` เพื่อขอสิทธิ์ในการอ่านสถานะของโทรศัพท์จากแอนดรอยด์

```
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

3 เพิ่มการประกาศ BroadcastReceiver ที่เราสร้างขึ้น (คลาส MyBroadcastReceiver) ในไฟล์ AndroidManifest.xml โดยระบุว่า BroadcastReceiver นี้สามารถจัดการอินเทนต์ที่เป็นแอคชั่น android.intent.action.PHONE_STATE ได้

```
<application
    ... >
    <activity
        android:name="com.example.checkphonestatebackground.MainActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

    <receiver android:name=".MyBroadcastReceiver" >
        <intent-filter>
            <action android:name="android.intent.action.PHONE_STATE" />
        </intent-filter>
    </receiver>
</application>
```

ผลการรัน

หน้าจอของแอพนี้จะแสดงข้อความ Hello world! ตามที่ Eclipse ทำไว้ให้ตั้งแต่แรก เนื่องจากเราไม่ได้กำหนด Layout เอง และไม่ได้เขียนโค้ดในแอคทิวิตีแต่อย่างใด

เมื่อคุณสั่งรัน แอพก็จะถูกติดตั้งลงในอีมูเลเตอร์ และ BroadcastReceiver ของเราจะเริ่มทำงานทันที โดยทำงานอยู่หลังฉากตลอดเวลาไม่ว่าตัวแอพจะรันอยู่หรือไม่ก็ตาม ดังนั้นเมื่อแอพรันขึ้นมาแล้วคุณสามารถออกจากแอพได้เลย จากนั้นให้ลองโทรออกโดยใช้แอพ Phone ของแอนดรอยด์ ก็จะปรากฏ Toast ที่แจ้งสถานะการโทรเหมือนตัวอย่างที่แล้ว สำหรับสายเรียกเข้าก็จะได้ผลเหมือนตัวอย่างที่แล้วเช่นกัน

การปิดกั้นการโทรออกไปยังหมายเลขหนึ่งๆ

เราสามารถสร้างแอพที่ป้องกันไม่ให้ผู้ใช้โทรออกไปยังหมายเลขหนึ่งๆได้ ซึ่งมีประโยชน์สำหรับผู้ปกครองที่จะควบคุมการใช้งานสมาร์ทโฟนหรือแท็บเล็ตของบุตรหลาน เช่น ผู้ปกครองอาจตั้งค่าไม่ให้โทรออกไปยังหมายเลขอื่นๆได้นอกจากหมายเลขโทรศัพท์ของผู้ปกครองหรือหมายเลขโทรศัพท์ที่บ้านเท่านั้น

ตัวอย่างและคำอธิบาย

ตัวอย่างนี้จะใช้ BroadcastReceiver คอยตรวจสอบการโทรออก และจะยกเลิกการโทรหากพบว่าหมายเลขที่จะโทรออกไปคือ 12345

1 สร้างคลาสใหม่ชื่อ BlockCallReceiver แล้วพิมพ์โค้ดดังนี้

โปรเจ็ค BlockOutgoingCall, ไฟล์ BlockCallReceiver.java

```
package com.example.blockoutgoingcall;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;

public class BlockCallReceiver extends BroadcastReceiver {

    Context context;

    @Override
    public void onReceive(Context context, Intent intent) {
        this.context = context;

        String outgoingNumber =
                intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER); ❶

        if (outgoingNumber.equals("12345")) { ❷
            setResultData(null); ❸
            showToast("This call is not allowed!");
        }
    }

    private void showToast(String msg) {
        Toast.makeText(context, msg, Toast.LENGTH_SHORT).show();
    }
}
```

หมายเลขโทรออกจะเป็นข้อมูลในอินเทนต์ที่แอนดรอยด์ broadcast ออกมา เราอ่านข้อมูลนี้โดยใช้เมธอด `getStringExtra` และระบุคีย์ของข้อมูลเป็น `Intent.EXTRA_PHONE_NUMBER` ❶

ถ้าหากหมายเลขโทรออกคือ `12345` ❷ ก็จะแก้ไขข้อมูลในการ broadcast ครั้งนั้นเป็นค่า `null` ซึ่งมีผลให้การโทรออกถูกยกเลิกไป ❸

2 เพิ่มบรรทัดต่อไปนี้ในไฟล์ AndroidManifest.xml โดยใส่ไว้ก่อนแท็ก `<application>` เพื่อขอสิทธิ์จัดการเรื่องการโทรออกจากแอนดรอยด์

```
<uses-permission android:name="android.permission.PROCESS_OUTGOING_CALLS" />
```

3 เพิ่มการประกาศ `BroadcastReceiver` ที่เราสร้างขึ้น (คลาส `BlockCallReceiver`) ในไฟล์ AndroidManifest.xml โดยระบุว่า `BroadcastReceiver` นี้ สามารถจัดการอินเทนต์ที่เป็นแอคชั่น `android.intent.action.NEW_OUTGOING_CALL` ได้

```
<application
    ... >
    <activity
        android:name="com.example.blockoutgoingcall.MainActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

    <receiver android:name=".BlockCallReceiver" >
       <intent-filter android:priority="0" >
          <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
       </intent-filter>
    </receiver>
</application>
```

ผลการรัน

เมื่อรันแอพแล้วสามารถออกจากแอพได้เลย จากนั้นให้เปิดแอพ Phone แล้วโทรไปยังหมายเลข 12345 จะปรากฏ Toast ที่แสดงข้อความว่า This call is not allowed! และแอพ Phone จะถูกปิดลงไป แต่หากโทรหมายเลขอื่นจะสามารถโทรได้ตามปกติ

การรับสายอัตโนมัติ

เราสามารถสร้างแอพที่รับสายเรียกเข้าอัตโนมัติ โดยอาจให้ผู้ใช้กำหนดได้ว่าจะรับสายอัตโนมัติตอนไหนและเฉพาะหมายเลขใดบ้าง ซึ่งเป็นการอำนวยความสะดวกแก่ผู้ใช้ในช่วงเวลาที่ไม่สะดวกจะกดรับสายเอง เช่น ขณะขับรถอยู่และได้ต่อหูฟังบลูทูธไว้แล้ว เป็นต้น

ตัวอย่างและคำอธิบาย

ตัวอย่างนี้จะรับสายอัตโนมัติถ้าหากหมายเลขที่โทรเข้ามาคือ 7890 (ในทางปฏิบัติจริงๆอาจทำหน้าจอให้ผู้ใช้ตั้งค่าว่าจะรับสายอัตโนมัติเมื่อใด เช่น เมื่อมีหูฟังบลูทูธต่ออยู่ และให้กำหนดได้ว่าหมายเลขใดบ้างที่จะรับสายอัตโนมัติ แต่ในที่นี้จะแสดงโค้ดที่ใช้รับสายอัตโนมัติเท่านั้น)

1 สร้างคลาสใหม่ชื่อ AutoAnswerReceiver แล้วพิมพ์โค้ดดังนี้

โปรเจ็ค AutoAnswerCall, ไฟล์ AutoAnswerReceiver.java

```
package com.example.autoanswercall;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.telephony.TelephonyManager;
import android.view.KeyEvent;

public class AutoAnswerReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        String state = intent.getStringExtra(
                            TelephonyManager.EXTRA_STATE); ❶

        if (state.equals(TelephonyManager.EXTRA_STATE_RINGING)) { ❷
            String incomingNumber = intent.getStringExtra(
                        TelephonyManager.EXTRA_INCOMING_NUMBER); ❸
            if (incomingNumber.equals("7890")) { ❹
                Intent i = new Intent(Intent.ACTION_MEDIA_BUTTON); ❺

                i.putExtra(Intent.EXTRA_KEY_EVENT,
                           new KeyEvent(KeyEvent.ACTION_UP,
                                        KeyEvent.KEYCODE_HEADSETHOOK)); ❻

                context.sendOrderedBroadcast(i, null); ❼
            }
        }
    }
}
```

เราอ่านสถานะการโทรจากอินเทนต์ที่ส่งเป็นพารามิเตอร์เข้ามา ❶ ถ้าพบว่าสถานะคือ Ringing ❷ จะอ่านหมายเลขเรียกเข้าจากอินเทนต์ ❸ แล้วดูว่าเป็นหมายเลข 7890 หรือไม่ ❹ ถ้าใช่ก็จะรับสายอัตโนมัติ

การรับสายอัตโนมัติจะใช้วิธีจำลองการกดปุ่มรับสายที่หูฟัง (headset) โดยสร้างอินเทนต์ที่เป็นแอคชั่น ACTION_MEDIA_BUTTON ขึ้นมา ❺ แล้วใส่ข้อมูลลงในอินเทนต์ที่เป็นการบอกว่ามีการกดปุ่มที่หูฟังเพื่อรับสาย ❻ แล้วจึง broadcast อินเทนต์นั้นออกไป ❼

2 เพิ่มบรรทัดต่อไปนี้ในไฟล์ AndroidManifest.xml โดยใส่ไว้ก่อนแท็ก <application> เพื่อขอสิทธิ์ในการอ่านสถานะของโทรศัพท์จากแอนดรอยด์

```
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

3 เพิ่มการประกาศ BroadcastReceiver ที่เราสร้างขึ้น (คลาส AutoAnswerReceiver) ในไฟล์ AndroidManifest.xml โดยระบุว่า BroadcastReceiver นี้ สามารถจัดการอินเทนต์ที่เป็นแอคชั่น android.intent.action.PHONE_STATE ได้

```
<application
    ... >
    <activity
        android:name="com.example.autoanswercall.MainActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

    <receiver android:name=".AutoAnswerReceiver" >
        <intent-filter>
            <action android:name="android.intent.action.PHONE_STATE" />
        </intent-filter>
    </receiver>
</application>
```

ผลการรัน

เมื่อรันแอพแล้ว ให้ใช้แท็บ Emulator Control ใน DDMS Perspective จำลองสายเรียกเข้าโดยกรอกหมายเลขเรียกเข้าเป็น 7890 ซึ่งที่อีมูเลเตอร์จะปรากฏหน้าจอโทรศัพท์ที่บอกว่ามีสายเรียกเข้า จากนั้นสายจะถูกรับโดยอัตโนมัติ (ถ้าหาก BroadcastReceiver ในตัวอย่างCheckPhoneState Background ยังคงทำงานอยู่ จะปรากฏ Toast แจ้งสถานะ Ringing แล้วตามด้วยสถานะ Offhook ทันที)

การหาหมายเลขโทรศัพท์, รหัสซิมการ์ด และหมายเลข IMEI ของเครื่อง

การรู้หมายเลขโทรศัพท์, รหัสซิมการ์ด และหมายเลข IMEI ของเครื่อง จะช่วยให้เราสร้างแอพที่น่าสนใจได้ ยกตัวอย่างเช่น แอพของเราอาจบันทึกรหัสซิมการ์ดของเครื่องเก็บไว้ จากนั้นทุกครั้งที่มีการบูตเครื่องใหม่ก็อ่านรหัสซิมการ์ดมาเปรียบเทียบกับรหัสที่บันทึกไว้ เพื่อดูว่ามีการเปลี่ยนซิมหรือไม่ ถ้ามีการเปลี่ยนซิมก็จะส่ง SMS ไปยังหมายเลขหนึ่งๆโดยอัตโนมัติ เป็นต้น

NOTE

- IMEI (International Mobile Equipment Identity) คือหมายเลขประจำตัวของเครื่อง
- รหัสซิมการ์ด (SIM Card ID) คือรหัสประจำตัวของซิมการ์ด
- หมายเลขโทรศัพท์ คือหมายเลขโทรศัพท์ที่ผูกกับซิมการ์ด ซึ่งอาจเปลี่ยนแปลงได้

ตัวอย่างและคำอธิบาย

ตัวอย่างนี้จะอ่านข้อมูลทั้งสามดังกล่าว รวมถึงข้อมูลอื่นๆที่น่าสนใจ มาแสดงในหน้าจอของแอพ

1 กำหนด Layout ของหน้าจอ

โปรเจ็ค GetPhoneInfo, ไฟล์ activity__main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Phone number:" />

    <TextView
        android:id="@+id/phone_number"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#ccff99"
        android:padding="5dp"
        android:text="" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:text="SIM card ID:" />
```

```
    <TextView
        android:id="@+id/sim_id"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#ccff99"
        android:padding="5dp"
        android:text="" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:text="IMEI number:" />

    <TextView
        android:id="@+id/imei_number"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#ccff99"
        android:padding="5dp"
        android:text="" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:text="Other information:" />

    <TextView
        android:id="@+id/other_info"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#ccff99"
        android:padding="5dp"
        android:text="" />

</LinearLayout>
```

2 เพิ่มโค้ดในเมธอด onCreate ของแอคทิวิตี

โปรเจ็ค GetPhoneInfo, ไฟล์ MainActivity.java

```
TextView txtPhoneNumber = (TextView) findViewById(R.id.phone_number);
TextView txtSimId = (TextView) findViewById(R.id.sim_id);
TextView txtImeiNumber = (TextView) findViewById(R.id.imei_number);
TextView txtOtherInfo = (TextView) findViewById(R.id.other_info);

TelephonyManager telephony = (TelephonyManager)
                             getSystemService(Context.TELEPHONY_SERVICE);
// หมายเลขโทรศัพท์
String phoneNumber = telephony.getLine1Number();
if (phoneNumber != null) {
    txtPhoneNumber.setText(phoneNumber);
}
// รหัสซิมการ์ด
String simId = telephony.getSimSerialNumber();
if (simId != null) {
    txtSimId.setText(simId);
}
// หมายเลข IMEI
String imeiNumber = telephony.getDeviceId();
if (imeiNumber != null) {
    txtImeiNumber.setText(imeiNumber);
}
// ข้อมูลอื่นๆ
String text = "Operator name: " + telephony.getNetworkOperatorName() + "\n";
text += "SIM operator name: " + telephony.getSimOperatorName();
txtOtherInfo.setText(text);
```

การหาข้อมูลต่างๆจะทำได้โดยเรียกเมธอดของ TelephonyManager ซึ่งไม่มีอะไรยุ่งยาก จึงไม่ขออธิบายเพิ่ม

3 เพิ่มบรรทัดต่อไปนี้ในไฟล์ AndroidManifest.xml โดยใส่ไว้ก่อนแท็ก <application>

```
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

ผลการรัน

NOTE

ในบางกรณี การรันบนเครื่องจริงจะไม่สามารถอ่านหมายเลขโทรศัพท์จากซิมการ์ดได้ โดยเมธอด `getLine1Number` จะให้ค่าเป็นสตริงว่าง ดังนั้นหากคุณต้องการรู้หมายเลขโทรศัพท์อาจให้ผู้ใช้กรอกเข้ามาเอง และอาจให้ยืนยันโดยแอพของคุณติดต่อไปยังเซิร์ฟเวอร์เพื่อให้ส่ง SMS ไปที่หมายเลขนั้น แล้วผู้ใช้นำรหัสจาก SMS ไปกรอกลงในแอพเป็นต้น

การส่ง SMS โดยใช้แอพที่มีอยู่ในเครื่อง

ในหัวข้อนี้จะแสดงตัวอย่างการส่ง SMS โดยใช้แอพที่มีอยู่แล้วในเครื่อง ซึ่งปกติจะเป็นแอพชื่อ Messages หรือ Messaging แต่อาจเป็นแอพตัวอื่นที่ผู้ใช้ติดตั้งเพิ่มเองก็ได้

ตัวอย่าง

หน้าจอของแอพนี้จะมีปุ่มเพียงปุ่มเดียว ซึ่งเมื่อคลิกจะเรียกแอพสำหรับส่ง SMS ขึ้นมา โดยที่เราจะใส่เนื้อหาของ SMS และหมายเลขผู้รับไว้ให้เรียบร้อย

1 กำหนด Layout ของหน้าจอ

โปรเจ็ค SendSmsByApp, ไฟล์ activity_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp" >

    <Button
        android:id="@+id/run_app_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:text="Run app to send SMS" />

</LinearLayout>
```

2 เขียนโค้ดในเมธอด onCreate ของแอคทิวิตี

โปรเจ็ค SendSmsByApp, ไฟล์ MainActivity.java

```
Button button = (Button) findViewById(R.id.run_app_button);
button.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View v) {
        Intent intent = new Intent(android.content.Intent.ACTION_VIEW);
        intent.setType("vnd.android-dir/mms-sms");
        intent.putExtra("address", "5556");
        intent.putExtra("sms_body", "Hello!\nHow are you?");
        startActivity(intent);
    }
});
```

ผลการรัน

แอพของเรา แอพสำหรับส่ง SMS ที่มีอยู่แล้วในเครื่อง

คำอธิบาย

การเรียกแอพสำหรับส่ง SMS จะทำได้โดยสร้างอินเทนต์ที่เป็นแอคชั่น ACTION_VIEW และกำหนดชนิดข้อมูลของอินเทนต์ (MIME Data Type) เป็น vnd.android-dir/mms-sms

```
Intent intent = new Intent(android.content.Intent.ACTION_VIEW);
intent.setType("vnd.android-dir/mms-sms");
```

สำหรับหมายเลขผู้รับและเนื้อหาของ SMS ให้ใส่ลงในอินเทนต์โดยระบุคีย์เป็น address และ sms_body ตามลำดับ

```
intent.putExtra("address", "5556");
intent.putExtra("sms_body", "Hello!\nHow are you?");
```

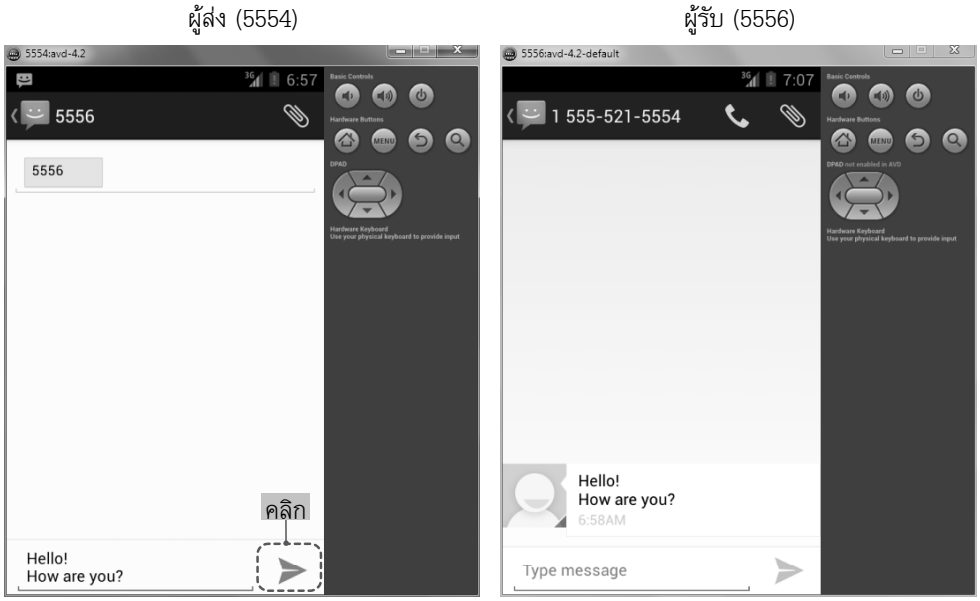
เสร็จแล้วให้เรียกเมธอด startActivity โดยระบุอินเทนต์นั้นเป็นพารามิเตอร์ แอนดรอยด์ก็จะเรียกแอพที่ใช้ส่ง SMS ขึ้นมา

```
startActivity(intent);
```

ทดสอบการส่ง SMS ระหว่างอีมูเลเตอร์

อีมูเลเตอร์แต่ละตัวหรือแต่ละอินสแทนซ์ (Instance) ที่รันอยู่จะมีหมายเลขพอร์ต (Port Number) ประจำตัว โดยเริ่มจาก 5554 สำหรับอีมูเลเตอร์ตัวแรก, 5556 สำหรับอีมูเลเตอร์ตัวที่สอง และเพิ่มขึ้นทีละ 2 สำหรับอีมูเลเตอร์ที่รันขึ้นมาถัดๆไป ซึ่งหมายเลขพอร์ตจะแสดงอยู่บน Title bar ของอีมูเลเตอร์ ดังรูป

เราสามารถทดสอบการโทรหรือรับ-ส่ง SMS ระหว่างอีมูเลเตอร์ที่รันอยู่โดยใช้หมายเลขพอร์ตดังกล่าวแทนหมายเลขโทรศัพท์จริงๆ เช่น สำหรับตัวอย่างนี้ให้คุณรันอีมูเลเตอร์ขึ้นมาอีกตัวหนึ่ง (หมายเลขพอร์ตจะเป็น 5556) จากนั้นไปที่อีมูเลเตอร์ตัวแรก (5554) ที่รันแอพในตัวอย่างนี้อยู่ (ถ้าปิดแอพไปแล้วก็ให้รันใหม่) แล้วทดลองส่ง SMS ก็จะทำให้ SMS นั้นถูกส่งไปยังอีมูเลเตอร์ 5556 ดังรูป



การส่ง SMS จากแอพของเราเอง

ตัวอย่างที่แล้วเป็นการใช้อินเทนต์เรียกแอพที่ใช้ส่ง SMS ขึ้นมา แล้วให้ผู้ใช้คลิกส่งเอง แต่ที่จริงเราสามารถเขียนโค้ดเพื่อให้ส่ง SMS จากแอพของเราได้โดยตรง ดังที่จะอธิบายในหัวข้อนี้

ตัวอย่างและคำอธิบาย

ตัวอย่างนี้จะมีหน้าจอให้กรอกหมายเลขโทรศัพท์ของผู้รับและข้อความที่ต้องการส่ง จากนั้นเมื่อคลิกปุ่มก็จะส่ง SMS ออกไป

1 กำหนด Layout ของหน้าจอ

โปรเจ็ค SendSms, ไฟล์ activity__main.xml

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp" >
```

```
    <TableRow
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="To:" />

        <EditText
            android:id="@+id/phone_number"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:ems="10"
            android:inputType="phone" >

            <requestFocus />
        </EditText>
    </TableRow>

    <TableRow
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp" >

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Text:" />

        <EditText
            android:id="@+id/message"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:ems="10" />
    </TableRow>

    <Button
        android:id="@+id/send_sms_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:text="Send SMS" />

</TableLayout>
```

2 เพิ่มโค้ดในเมธอด onCreate ของแอคทิวิตี

โปรเจ็ค SendSms, ไฟล์ MainActivity.java

```
final EditText etPhoneNumber = (EditText) findViewById(R.id.phone_number);
final EditText etMessage = (EditText) findViewById(R.id.message);

Button btnSendSms = (Button) findViewById(R.id.send_sms_button);
btnSendSms.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View v) {
        String phoneNumber = etPhoneNumber.getText().toString();
        String message = etMessage.getText().toString();

        SmsManager sms = SmsManager.getDefault(); ❶
        sms.sendTextMessage(phoneNumber, null, message, null, null); ❷
    }
});
```

การส่ง SMS นั้นก่อนอื่นต้องเรียกเมธอด getDefault ของคลาส SmsManager เพื่อเข้าถึงตัวจัดการ SMS (ออบเจ็ค SmsManager) ของระบบ ❶

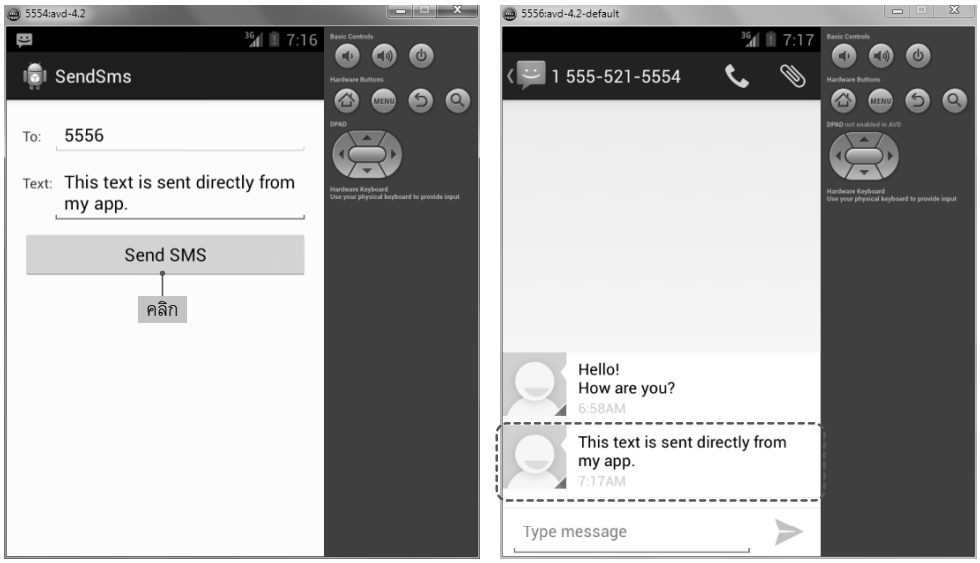
หลังจากนั้นให้ใช้เมธอด sendTextMessage ในการส่ง SMS โดยระบุหมายเลขโทรศัพท์ของผู้รับเป็นพารามิเตอร์ตัวแรก และระบุข้อความที่จะส่งเป็นพารามิเตอร์ตัวที่ 3 ❷

3 เพิ่มบรรทัดต่อไปนี้ในไฟล์ AndroidManifest.xml โดยใส่ไว้ก่อนแท็ก <application> เพื่อขอสิทธิ์ในการส่ง SMS

```
<uses-permission android:name="android.permission.SEND_SMS" />
```

ผลการรัน

ให้รันอีมูเลเตอร์ขึ้นมา 2 ตัว จากนั้นสั่งรันแอพบนอีมูเลเตอร์ 5554 แล้วทดลองส่ง SMS ไปยังอีมูเลเตอร์ 5556 ดังรูปหน้าถัดไป



การตรวจสอบสถานะของ SMS ที่ส่งออกไป

แอนดรอยด์มีวิธีให้เราตรวจสอบว่า SMS ที่ส่งออกไปนั้นส่งสำเร็จหรือไม่ และผู้รับได้รับหรือไม่ ทั้งนี้การส่งสำเร็จไม่ได้หมายความว่าผู้รับได้รับข้อความแล้ว เนื่องจากผู้รับอาจปิดเครื่องอยู่ก็ได้ ส่วนกรณีที่ส่งไม่สำเร็จก็เช่น ระบุหมายเลขโทรศัพท์ที่มีรูปแบบไม่ถูกต้อง หรือขณะนั้นไม่มีสัญญาณโทรศัพท์ เป็นต้น

ตัวอย่างและคำอธิบาย

ตัวอย่างนี้จะเขียนโค้ดต่อจากตัวอย่างที่แล้ว โดยเพิ่มการตรวจสอบสถานะของ SMS ที่ส่งออกไป (ในซอร์สโค้ดจะแยกออกมาเป็นโปรเจ็คใหม่ชื่อ SendSmsGetFeedback)

โค้ดที่ใช้ตรวจสอบสถานะของ SMS อาจดูยุ่งยากสักหน่อย หลักการคร่าวๆคือ เราจะเตรียมออบเจ็ค `PendingIntent` ให้แก่เมธอด `sendTextMessage` เพื่อให้ตัวจัดการ SMS ทำการ broadcast `PendingIntent` นั้นออกมาเพื่อแจ้งถึงความสำเร็จหรือล้มเหลวของการส่ง SMS โดยเราต้องเตรียม `BroadcastReceiver` ที่คอยรับ `PendingIntent` นั้นมาตรวจสอบด้วย ซึ่งทั้ง `PendingIntent` และ `BroadcastReceiver` นี้ต้องใช้ 2 ชุด ชุดหนึ่งสำหรับตรวจสอบสถานะการส่ง และอีกชุดหนึ่งสำหรับตรวจสอบสถานะการรับของผู้รับ

1 ที่แอคทิวิตี ให้เพิ่มการประกาศตัวแปรระดับคลาส

โปรเจ็ค SendSmsGetFeedback, ไฟล์ MainActivity.java

```
final String SENT = "message_sent";
final String RECEIVED = "message_received";

PendingIntent sentIntent, receivedIntent;
BroadcastReceiver sentReceiver, receivedReceiver;
```

2 เพิ่มโค้ดในเมธอด onCreate โดยใส่ไว้ถัดจากบรรทัด setContentView

โปรเจ็ค SendSmsGetFeedback, ไฟล์ MainActivity.java

```
sentIntent = PendingIntent.getBroadcast(this, 0, new Intent(SENT), 0);
receivedIntent = PendingIntent.getBroadcast(this, 0,
                                            new Intent(RECEIVED), 0);
```

คือการสร้าง PendingIntent ที่จะส่งให้เมธอด sendTextMessage และให้สังเกตว่ามีการสร้างอินเทนต์ขึ้นมาภายใน PendingIntent ทั้งสอง โดยระบุแอคชั่นเป็นค่าคงที่ SENT และ RECEIVED ซึ่งอินเทนต์นี้เองที่ตัวจัดการ SMS จะ broadcast ออกมา เพื่อแจ้งผลกลับมายังโค้ดของเรา

3 ในเมธอด onClick ที่กำหนดการทำงานของปุ่ม ให้แก้ไขเมธอด sendTextMessage โดยนำ PendingIntent มาระบุเป็นพารามิเตอร์ตัวที่ 4 และ 5 ของเมธอด

โปรเจ็ค SendSmsGetFeedback, ไฟล์ MainActivity.java

```
sms.sendTextMessage(phoneNumber, null, message, sentIntent, receivedIntent);
```

4 เพิ่มเมธอด onResume ในแอคทิวิตี

โปรเจ็ค SendSmsGetFeedback, ไฟล์ MainActivity.java

```
@Override
protected void onResume() {
    super.onResume();

    // BroadcastReceiver ที่ตรวจสอบสถานะการส่ง
    sentReceiver = new BroadcastReceiver() {

        @Override
        public void onReceive(Context context, Intent intent) {
            switch (getResultCode()) {
            case Activity.RESULT_OK:
                showToast("Message sent");
                break;
            case SmsManager.RESULT_ERROR_GENERIC_FAILURE:
                showToast("Message not sent: Generic failure");
                break;
```

```
            case SmsManager.RESULT_ERROR_NO_SERVICE:
                showToast("Message not sent: No service");
                break;
            case SmsManager.RESULT_ERROR_NULL_PDU:
                showToast("Message not sent: Null PDU");
                break;
            case SmsManager.RESULT_ERROR_RADIO_OFF:
                showToast("Message not sent: Radio off");
                break;
            }
        }
    };

    // BroadcastReceiver ที่ตรวจสอบสถานะการรับของผู้รับ
    receivedReceiver = new BroadcastReceiver() {

        @Override
        public void onReceive(Context context, Intent intent) {
            switch (getResultCode()) {
            case Activity.RESULT_OK:
                showToast("Message received");
                break;
            case Activity.RESULT_CANCELED:
                showToast("Message not received");
                break;
            }
        }
    };

    /* ลงทะเบียน BroadcastReceiver ทั้งสอง โดยระบุว่าต้องการจัดการกับอินเทนต์ที่มีแอคชั่น
       เป็นค่าคงที่ SENT และ RECEIVED (ซึ่งก็คืออินเทนต์ที่เราเตรียมไว้ให้ตัวจัดการ SMS โดยผ่านทาง
       เมธอด sendTextMessage) */
    registerReceiver(sentReceiver, new IntentFilter(SENT));
    registerReceiver(receivedReceiver, new IntentFilter(RECEIVED));
}
```

เราสร้าง BroadcastReceiver ขึ้นมา 2 ออบเจ็ค เพื่อทำหน้าที่ตรวจสอบสถานะการส่งและสถานะการรับของผู้รับ แล้วนำออบเจ็คทั้งสองมาลงทะเบียนกับระบบ

สาเหตุที่ใส่โค้ดข้างต้นไว้ในเมธอด onResume ของแอคทิวิตี ก็เพราะเราต้องการตรวจสอบสถานะ SMS เฉพาะตอนที่แอพของเรารันอยู่เบื้องหน้าเท่านั้น

5 เพิ่มเมธอด onPause ในแอคทิวิตี

โปรเจ็ค SendSmsGetFeedback, ไฟล์ MainActivity.java

```
@Override
protected void onPause() {
    super.onPause();

    unregisterReceiver(sentReceiver);
    unregisterReceiver(receivedReceiver);
}
```

เราจะยกเลิกการลงทะเบียน BroadcastReceiver ทั้งสอง (ยกเลิกการตรวจสอบสถานะ SMS) เมื่อแอพของเรากำลังจะลงไปอยู่เบื้องหลัง เช่น เมื่อมีแอพอื่นถูกรันขึ้นมาเบื้องหน้าแทน หรือผู้ใช้กำลังออกจากแอพของเรา เป็นต้น

6 เพิ่มเมธอด showToast ในแอคทิวิตี

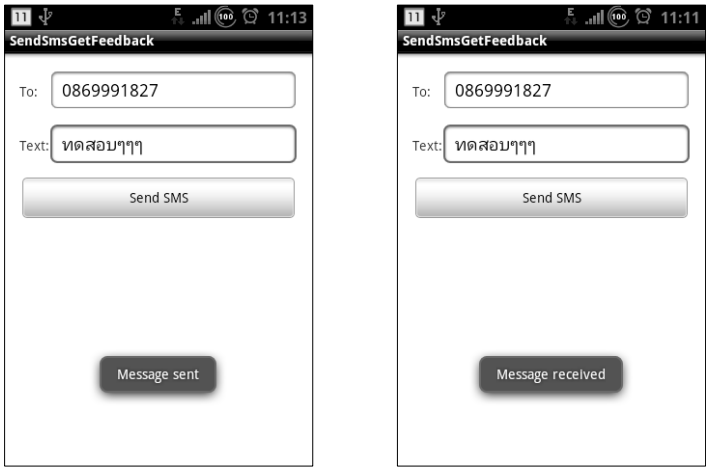
โปรเจ็ค SendSmsGetFeedback, ไฟล์ MainActivity.java

```
private void showToast(String msg) {
    Toast.makeText(this, msg, Toast.LENGTH_SHORT).show();
}
```

ผลการรัน

สั่งรันแอพบนอีมูเลเตอร์ 5554 แล้วทดลองส่ง SMS ไปยังอีมูเลเตอร์ 5556 หลังจากส่งแล้วจะมี Toast ที่บอกสถานะการส่งปรากฏขึ้นมาในหน้าจอของอีมูเลเตอร์ 5554 ดังรูป

การทดสอบบนอีมูเลเตอร์จะไม่มีการแจ้งสถานะการรับของผู้รับกลับมา แต่บนเครื่องจริงจะมีดังรูป



การตรวจสอบ SMS ที่ส่งออกไปจากเครื่อง

เราสามารถกำหนดให้แอนดรอยด์แจ้งมายังแอพของเราทุกครั้งที่มีการส่ง SMS ออกไปโดยใช้แอพ Messaging ที่มากับเครื่องได้

ตัวอย่างและคำอธิบาย

ข้อความ SMS ที่ส่งออกไปโดยแอพ Messaging นั้นจะถูกจัดเก็บลงใน Content Provider ที่ตำแหน่ง `content://sms/sent` ซึ่งเราสามารถลงทะเบียนกับ Content Provider เพื่อให้แจ้งกลับมาเมื่อข้อมูลที่ตำแหน่งดังกล่าวเปลี่ยนแปลงไป

1 ที่แอคทิวิตี ให้เพิ่มการประกาศตัวแปรระดับคลาส

โปรเจ็ค MonitorOutgoingSMS, ไฟล์ MainActivity.java

```
MyContentObserver observer;
```

ตัวแปร `observer` จะใช้เก็บ (อ้างอิง) ออบเจ็คของคลาส `MyContentObserver` ซึ่งเป็นคลาสที่เราเขียนขึ้นเอง รายละเอียดของคลาสนี้จะอธิบายต่อไป

2 เพิ่มโค้ดในเมธอด `onCreate` ของแอคทิวิตี

โปรเจ็ค MonitorOutgoingSMS, ไฟล์ MainActivity.java

```
observer = new MyContentObserver(new Handler(), this);
Uri uri = Uri.parse("content://sms");
getContentResolver().registerContentObserver(uri, true, observer);
```

คือการสร้างออบเจ็คของคลาส MyContentObserver แล้วนำไปลงทะเบียนกับ Content Provider เพื่อคอยสังเกตการเปลี่ยนแปลงที่เกิดกับข้อมูล SMS ใน Content Provider ทั้งนี้เมื่อเกิดการเปลี่ยนแปลงขึ้น แอนดรอยด์จะเรียกมายังเมธอด onChange ในออบเจ็คของเรา

3 เพิ่มเมธอด onDestroy ในแอคทิวิตี

โปรเจ็ค MonitorOutgoingSMS, ไฟล์ MainActivity.java

```
@Override
protected void onDestroy() {
    super.onDestroy();
    getContentResolver().unregisterContentObserver(observer);
}
```

เมื่อแอคทิวิตีกำลังจะจบการทำงาน เราจะยกเลิกการลงทะเบียนตรวจสอบ SMS ดังกล่าว

4 สร้างคลาสใหม่ชื่อ MyContentObserver แล้วพิมพ์โค้ดดังนี้

โปรเจ็ค MyContentObserver, ไฟล์ MainActivity.java

```
package com.example.monitoroutgoingsms;

import android.content.Context;
import android.database.ContentObserver;
import android.database.Cursor;
import android.net.Uri;
import android.os.Handler;
import android.widget.Toast;

public class MyContentObserver extends ContentObserver {

    private Context context;
    private int smsCount;

    public MyContentObserver(Handler handler, Context context) {
        super(handler);
        this.context = context;

        // นับจำนวน SMS ที่ถูกส่งไปแล้วทั้งหมด เก็บไว้ในตัวแปร smsCount
        Uri SmsUri = Uri.parse("content://sms/sent");
        Cursor c = context.getContentResolver().query(SmsUri, null, null,
                null, null);
        smsCount = c.getCount();
        c.close();
    }
```

```
    @Override
    public void onChange(boolean selfChange) {
        Uri SmsUri = Uri.parse("content://sms/sent");
        String[] columns = { "address", "date", "body", "type" };

        Cursor c = context.getContentResolver().query(SmsUri, columns,
                                                    null, null, null); ❶
        c.moveToNext(); // ข้อมูล SMS ล่าสุดจะอยู่ในแถวข้อมูลถัดไป

        if (c.getCount() > smsCount) { ❷
            smsCount = c.getCount();

            String text = "Recipient: "
                        + c.getString(c.getColumnIndex(columns[0]));
            text += "\nDate: " + c.getString(c.getColumnIndex(columns[1]));
            text += "\nMessage: "
                    + c.getString(c.getColumnIndex(columns[2]));
            text += "\nType: " + c.getString(c.getColumnIndex(columns[3]));
            text += "\n--------------------";
            text += "\nSMS Count: " + String.valueOf(smsCount);

            Toast.makeText(context, text, Toast.LENGTH_LONG).show();
        }
        c.close();
    }

    @Override
    public boolean deliverSelfNotifications() {
        return true;
    }
}
```

เราสร้างคลาส `MyContentObserver` นี้ โดยสืบทอดจากคลาส `ContentObserver` ของแอนดรอยด์ แล้ว Override เมธอด `onChange` และเมธอด `deliverSelfNotifications`

เมธอด `onChange` คือตำแหน่งสำหรับระบุการทำงานเมื่อเกิดการเปลี่ยนแปลงกับข้อมูล SMS ใน Content Provider ซึ่งในที่นี้จะคิวรีข้อมูล SMS ที่ส่งไปแล้วทั้งหมด ❶ และอ่านข้อมูล SMS อันล่าสุดมาแสดงใน Toast

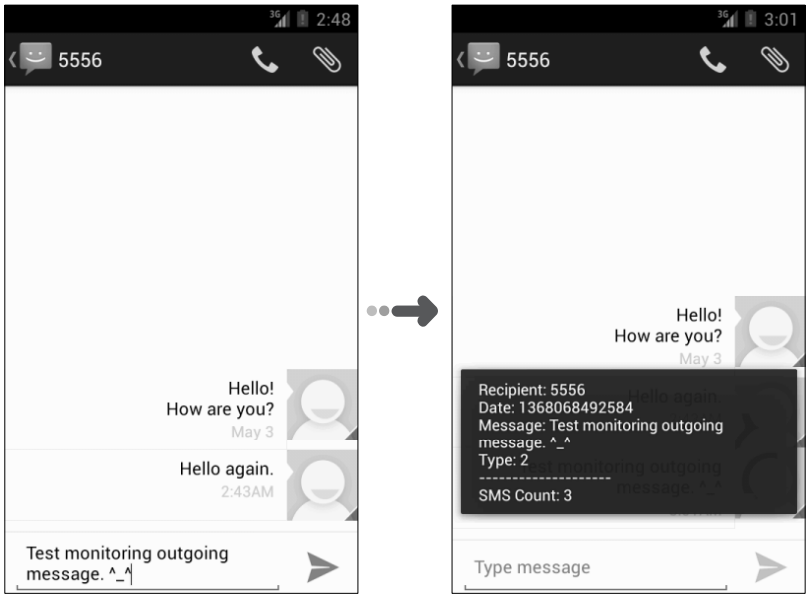
ข้อควรทราบคือ เมื่อมีการใช้แอพ Messaging ส่ง SMS ออกไป แอนดรอยด์จะเรียกมายังเมธอด onChange 3 ครั้ง โดย 2 ครั้งแรกสำหรับ SMS อันก่อนหน้านี้ และครั้งที่ 3 สำหรับ SMS อันล่าสุดที่เราต้องการ ดังนั้นในแต่ละครั้งที่เมธอด onChange ถูกเรียก เราจะนับจำนวน SMS ที่ส่งออกไปแล้วทั้งหมด เปรียบเทียบกับค่าที่บันทึกไว้ในตัวแปร smsCount ❷ ถ้ามากกว่าก็แสดงว่าการเรียกเมธอด onChange ในครั้งนั้นเป็นการแจ้งสำหรับ SMS อันล่าสุดที่ถูกส่งออกไป

5 เพิ่มบรรทัดต่อไปนี้ในไฟล์ AndroidManifest.xml โดยใส่ไว้ก่อนแท็ก <application> เพื่อขอสิทธิ์ในการอ่าน SMS

```
<uses-permission android:name="android.permission.READ_SMS" />
```

ผลการรัน

เมื่อรันแอพแล้วให้กด Home เพื่อไปที่หน้าโฮมสกรีน (อย่ากด Back เพราะจะทำให้แอพของเราจบการทำงาน ซึ่งเราได้ยกเลิกการลงทะเบียนตรวจสอบ SMS ไว้ในเมธอด onDestroy ของแอคทิวิตี) จากนั้นเปิดแอพ Messaging ขึ้นมา แล้วลองส่ง SMS ไปยังหมายเลขใดก็ได้ จะปรากฏ Toast ดังรูป



NOTE

ถ้าคุณจะนำตัวอย่างนี้ไปดัดแปลงใช้งานจริง ขอแนะนำให้สร้างเป็นเซอร์วิส (Service) แทนที่จะสร้างเป็นแอพธรรมดา เนื่องจากเซอร์วิสจะทำงานอยู่หลังฉากตลอดเวลาหลังจากผู้ใช้ติดตั้งแอพของคุณแล้ว

การรับ SMS

นอกจากการส่ง SMS แอนดรอยด์ยังมีวิธีให้เรารับ SMS ที่ส่งเข้ามาได้ด้วย ตัวอย่างการใช้งาน เช่น เราอาจให้ผู้ใช้รอรับรหัสผ่านจากเซิร์ฟเวอร์ที่เราจะส่งเป็น SMS ไปยังเครื่องโทรศัพท์ของผู้ใช้ ซึ่งเมื่อ SMS ส่งมาแล้วแอพของเราก็จะดักจับและอ่านรหัสผ่านมาใช้อัตโนมัติ โดยผู้ใช้ไม่ต้องเสียเวลาดูรหัสผ่านจาก SMS แล้วกรอกเอง ก็จะทำให้ผู้ใช้ได้รับประสบการณ์ที่ดีในการใช้งานแอพของเรา

ตัวอย่างและคำอธิบาย

เมื่อมี SMS ส่งเข้ามา แอนดรอยด์จะ broadcast อินเทนต์ `android.provider.Telephony.SMS_RECEIVED` ออกไป ดังนั้นเราจะสร้าง `BroadcastReceiver` ที่คอยรับอินเทนต์ดังกล่าว และระบุการทำงานที่ต้องการในเมธอด `onReceive` ของ `BroadcastReceiver` ซึ่งในตัวอย่างนี้จะแสดงหมายเลขโทรศัพท์ของผู้ส่งและข้อความ SMS ออกมาใน Toast

1 สร้างคลาสใหม่ชื่อ `SmsReceiver` แล้วพิมพ์โค้ดดังนี้

โปรเจ็ค ReceiveSMS, ไฟล์ SmsReceiver.java

```
package com.example.receivesms;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.telephony.SmsMessage;
import android.widget.Toast;

public class SmsReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        Bundle bundle = intent.getExtras();

        if (bundle != null) {
            // อ่านอาร์เรย์ของออบเจ็คจากอินเทนต์
            Object[] pdus = (Object[]) bundle.get("pdus");
            SmsMessage[] msgs = new SmsMessage[pdus.length];
            String str = "";

            for (int i = 0; i < msgs.length; i++) {
                /* สร้างออบเจ็ค SmsMessage ขึ้นมาจากออบเจ็คหนึ่งๆในอาร์เรย์ ซึ่งออบเจ็ค
                   SmsMessage นี้จะแทนข้อความส่วนหนึ่งใน SMS (ไม่เกิน 160 ตัวอักษร) */
                msgs[i] = SmsMessage.createFromPdu((byte[]) pdus[i]);
```

```
                // อ่านหมายเลขโทรศัพท์ของผู้ส่งจากออบเจ็คแรกในอาร์เรย์
                if (i == 0) {
                    str = "SMS from " + msgs[i].getOriginatingAddress();
                    str += "\n--------------------\n";
                }
                // อ่านข้อความแต่ละส่วนแล้วนำมาต่อกันเป็นข้อความ SMS ที่สมบูรณ์
                str += msgs[i].getMessageBody();
            }

            Toast.makeText(context, str, Toast.LENGTH_LONG).show();
        }
    }
}
```

เมธอด onReceive ในคลาส SmsReceiver ข้างต้นจะถูกเรียกให้ทำงานในแต่ละครั้งที่มี SMS ส่งเข้ามา เช่นถ้ามี SMS เข้ามา 3 ข้อความ เมธอดนี้ก็จะถูกเรียก 3 ครั้ง

ข้อความ SMS จะเก็บอยู่ในอินเทนต์ที่ส่งผ่านมายังพารามิเตอร์ตัวที่ 2 ของเมธอด onReceive โดยเป็นอาร์เรย์ของออบเจ็ค ซึ่งแต่ละออบเจ็คบรรจุข้อความไม่เกิน 160 ตัวอักษรในรูปแบบ PDU (Packet Data Unit) ดังนั้นหากข้อความ SMS ที่ส่งมามีมากกว่า 160 ตัวอักษรก็จะถูกแบ่งออกเป็นหลายส่วน โดยแต่ละส่วนคือ 1 ออบเจ็คในอาร์เรย์ดังกล่าว

2 เพิ่มบรรทัดต่อไปนี้ในไฟล์ AndroidManifest.xml โดยใส่ไว้ก่อนแท็ก <application> เพื่อขอสิทธิ์ในการรับ SMS

```
<uses-permission android:name="android.permission.RECEIVE_SMS" />
```

3 เพิ่มการประกาศ BroadcastReceiver ที่เราสร้างขึ้น (คลาส SmsReceiver) ในไฟล์ AndroidManifest.xml โดยระบุว่า BroadcastReceiver นี้สามารถจัดการอินเทนต์ android.provider.Telephony.SMS_RECEIVED ได้

```
<application
    ... >
    <activity
        android:name="com.example.receivesms.MainActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
```

```
    <receiver android:name=".SmsReceiver" >
        <intent-filter android:priority="100" >
            <action android:name="android.provider.Telephony.SMS_RECEIVED" />
        </intent-filter>
    </receiver>
</application>
```

ผลการรัน

รันแอพ แล้วจำลองการส่ง SMS ไปยังอีมูเลเตอร์โดยใช้แท็บ Emulator Control ใน DDMS Perspective ของ Eclipse