Android Developer Fundamentals

Phone and SMS

Lesson 1

1 Phone Calls

Contents

- Implicit intent: dialing vs. calling
- Formatting the phone number
- ACTION_DIAL to use Phone app to make call
- ACTION_CALL to call from within your app
- Using emulator instances to test phone calls

Implicit intent: dialing vs. calling

- ACTION_DIAL: Use the Phone app to dial the call
  - Pro: Simplest action without need for requesting user permission
  - Con: User navigates back to your app from the Phone app
- ACTION_CALL: Make the call from within your app
  - Pro: Keeps user within your app
  - Con: Need to request user permission

Phone number URI

Prepare a Uniform Resource Identifier (URI) as a string

- editText: EditText for entering a phone number (`14155551212`)
- phoneNumber: String prefixed by "tel:", e.g. (`tel:14155551212`)

```
String phoneNumber =
        String.format("tel: %s", editText.getText().toString());
```

PhoneNumberUtils class

PhoneNumberUtils class provides utility methods for phone number strings

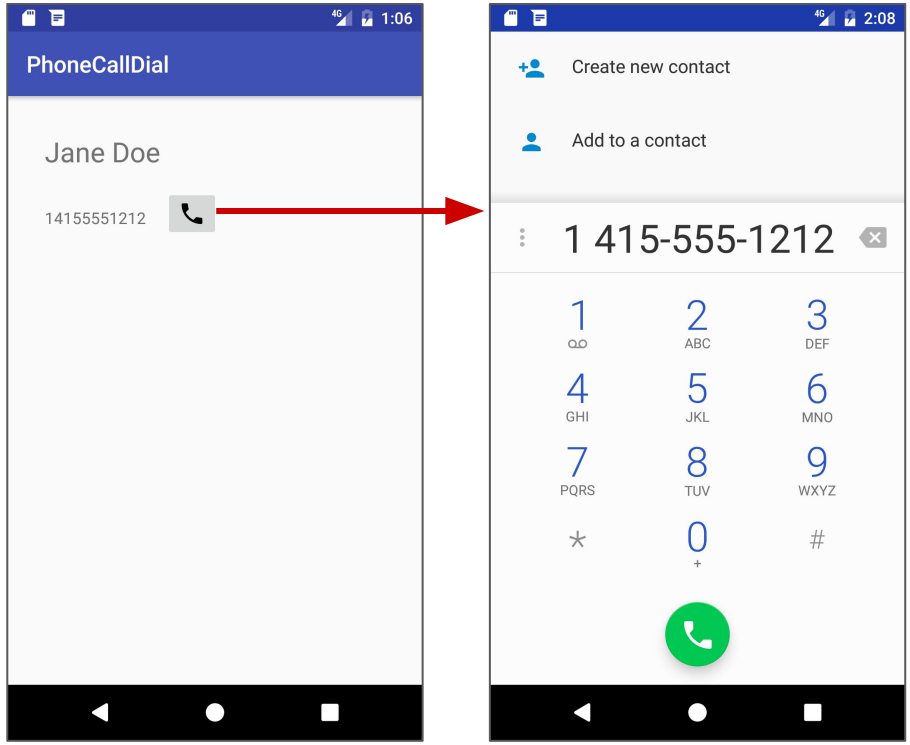
- normalizeNumber() removes characters other than digits
- formatNumber() formats a phone number for a specific country if the number doesn't already include a country code

Intent with ACTION_DIAL

Use the Phone app to dial a call

Intent with ACTION_DIAL uses Phone app to dial call

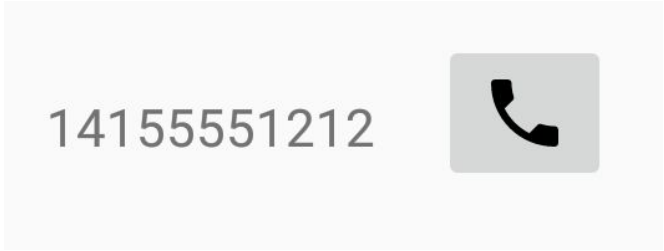
- User can change phone number before dialing the call
- User navigates back to your app by tapping **Back**



Add onClick handler for call button

1. Add call button and phone number
2. Add `android:onClick` attribute to button
3. Create handler for onClick

```
public void callNumber(View view) {
    ...
}
```



Create intent with phone number

1. Prepare a URI

```
String phoneNumber =
        String.format("tel: %s", editText.getText().toString());
```

2. Create implicit Intent

```
Intent dialIntent = new Intent(Intent.ACTION_DIAL);
```

3. setData() with phone number

```
dialIntent.setData(Uri.parse(phoneNumber));
```

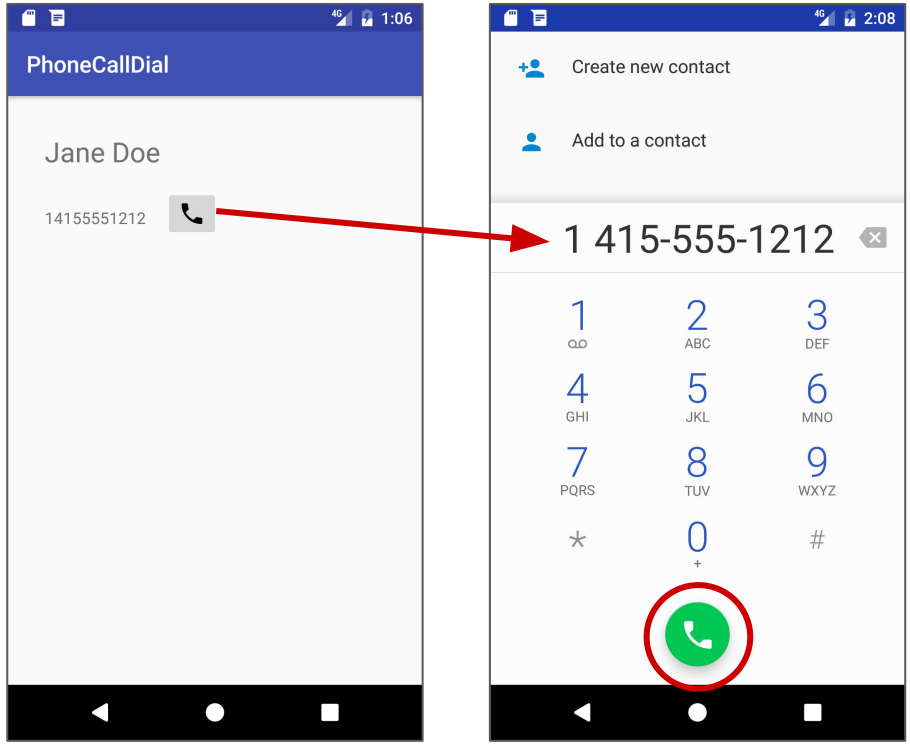
Send intent to Phone app

- Use resolveActivity() with getPackageManager() to resolve the intent to an installed app
- If result non-null, call `startActivity()`

```
if (dialIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(dialIntent);
} else {
    ... // Log and show explanation
```

Phone app dials and connects call

- Tap button to make call
- Phone number appears in Phone app
- Tap to dial and make call



Intent with ACTION_CALL

Using ACTION_CALL

1. Add permissions to AndroidManifest.xml

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

2. Check if telephony is enabled; if not, disable phone feature
3. Check if user still grants permission; request if needed
4. Extend PhoneStateListener; register listener using TelephonyManager
5. Create implicit intent

Check for TelephonyManager

- Telephony functionality is provided by TelephonyManager
- Not all devices have it, so you must check

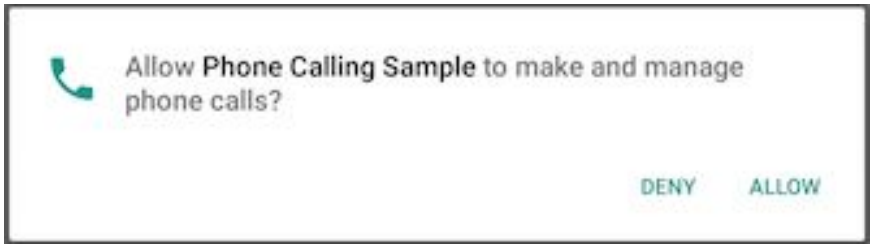
```
private Boolean isTelephonyEnabled() {
    TelephonyManager mTelephonyManager =
            (TelephonyManager) getSystemService(TELEPHONY_SERVICE);
    if (mTelephonyManager != null) {
        if (mTelephonyManager.getSimState() ==
            TelephonyManager.SIM_STATE_READY) {
            return true;
        } else { return false; }}
```

Is telephony enabled?

```
if (isTelephonyEnabled()) {
    // OK Telephony is enabled
} else {
  Toast.makeText(this,
     R.string.telephony_not_enabled,Toast.LENGTH_LONG).show();
  Log.d(TAG, getString(R.string.telephony_not_enabled));
  // Disable the call button
}
```

Do you have Phone permission?

- If the user turned off Phone permission for the app, your app can request permission



Check permission steps

- Define integer to use for the `requestCode` parameter

```
private static final int MY_PERMISSIONS_REQUEST_CALL_PHONE = 1;
```

- Use checkSelfPermission()
- Use requestPermissions()
  - Context (`this`), and `CALL_PHONE` in the string array of permissions
  - The request integer constant `MY_PERMISSIONS_REQUEST_CALL_PHONE`

Check permission code

```
if (ActivityCompat.checkSelfPermission(this,
    Manifest.permission.CALL_PHONE) !=
    PackageManager.PERMISSION_GRANTED) {

        ActivityCompat.requestPermissions(this,
                new String[]{Manifest.permission.CALL_PHONE},
                MY_PERMISSIONS_REQUEST_CALL_PHONE);
```

Get user's response

Override onRequestPermissionsResult() to check if returned `requestCode` is `MY_PERMISSIONS_REQUEST_CALL_PHONE`

```
@Override
public void onRequestPermissionsResult(int requestCode,
                        String permissions[], int[] grantResults) {
            switch (requestCode) {
                case MY_PERMISSIONS_REQUEST_CALL_PHONE: {
                    ...
```

Check if request was granted

- Response returned in `permissions` array (index 0 if only a single request was made)
- Compare to grant result: `PERMISSION_GRANTED` or `PERMISSION_DENIED`

```
if (permissions[0].equalsIgnoreCase(Manifest.permission.CALL_PHONE)
        && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
    // Permission was granted.
} else {
    ...
```

Extend PhoneStateListener

- PhoneStateListener monitors changes in telephony states
  - ringing, off-hook, idle
- Extend PhoneStateListener and override onCallStateChanged()

```
private class MyPhoneCallListener extends PhoneStateListener {
    @Override
    public void onCallStateChanged
                            (int state, String incomingNumber) {
    switch (state) {
    ...
```

Provide actions for phone states

Add actions you want to take based on the phone states

- State is CALL_STATE_IDLE state until a call is started
- State changes to CALL_STATE_OFFHOOK to make connection and remains in this state for the duration of the call
- State returns to CALL_STATE_IDLE after the call
- Your app resumes when the state changes back to CALL_STATE_IDLE
- State is CALL_STATE_RINGING when an incoming call is detected

CALL_STATE_RINGING example

```
switch (state) {
    case TelephonyManager.CALL_STATE_RINGING:
        // Incoming call is ringing, show number
        TextView incomingView = (TextView)
                                findViewById(R.id.incoming);
        incomingView.setText(incomingNumber);
        incomingView.setVisibility(View.VISIBLE);
        break;
    case TelephonyManager.CALL_STATE_OFFHOOK:
        // Phone call is active (phone off the hook)
        ...
```

Register listener

- Register listener in `onCreate()` method using telephonyManager.listen() with the PhoneStateListener set to `LISTEN_CALL_STATE`

```
@Override
    protected void onCreate(Bundle savedInstanceState) {
    ...
    MyPhoneCallListener mListener = new MyPhoneCallListener();
    telephonyManager.listen(mListener,
                        PhoneStateListener.LISTEN_CALL_STATE);
}
```

Send intent with phone number

1. Prepare a URI
2. Create the implicit intent and include setData() with phone number
3. Use resolveActivity() with getPackageManager() to resolve implicit intent to an installed app
4. If result non-null, call `startActivity()`

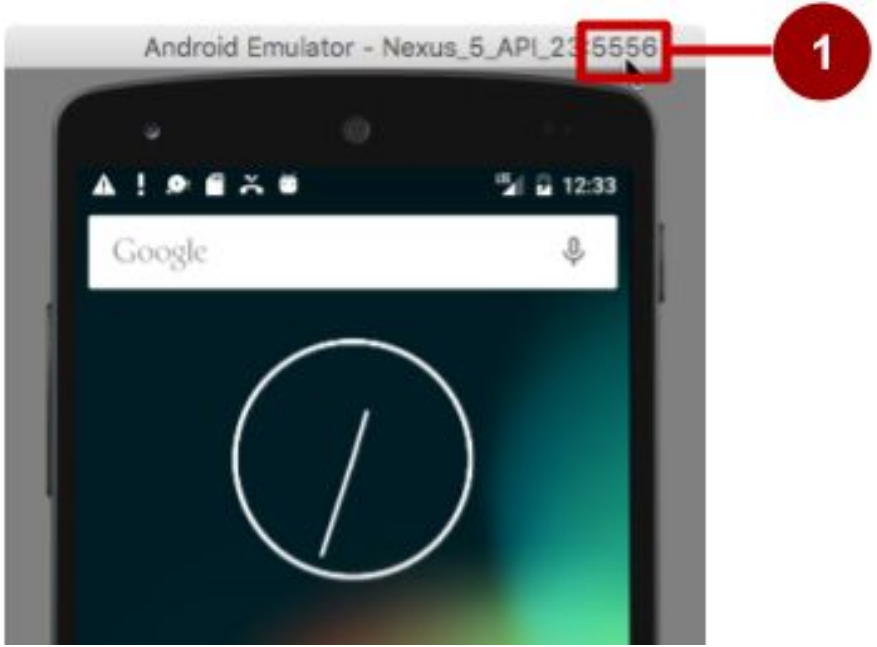
Code to send intent

```
String phoneNumber = String.format("tel: %s",
                                   editText.getText().toString());
Intent callIntent = new Intent(Intent.ACTION_CALL);
callIntent.setData(Uri.parse(phoneNumber));
if (callIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(callIntent);
} else {
    ... // Log and show explanation.
```

Using Two Emulators to Test Phone Calls

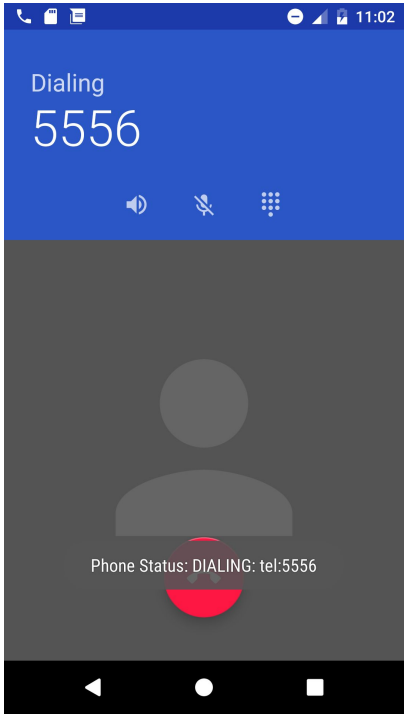
Launch first emulator

- Choose **Tools > Android > AVD Manager** and double-click a predefined device
- Note number in emulator window title on the far right (e.g. 5556)



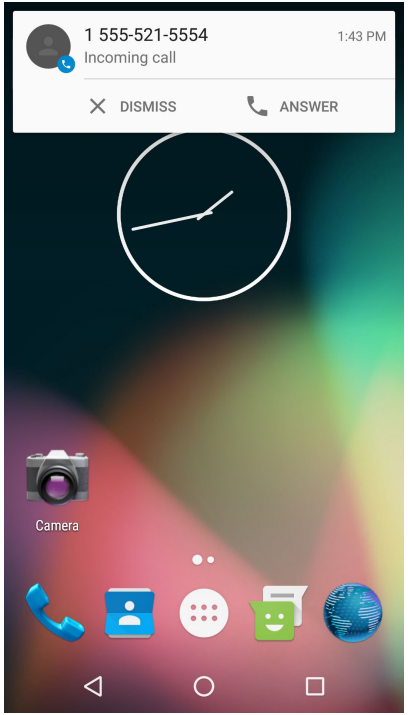
Run app on a second emulator

1. Launch second emulator
2. Run app on second emulator
3. Enter first emulator's number (e.g. 5556) as phone number to call
4. Complete calling



Receive call on first emulator

5. First emulator receives the call
6. Answer or dismiss the call



Learn more

- Common Intents and Intents and Intent Filters
- TelephonyManager
- PhoneStateListener
- Requesting Permissions at Run Time
- checkSelfPermission
- Run Apps on the Android Emulator

What's Next?

- Concept Chapter: Phone Calls
- Practical: Making Phone Calls

END