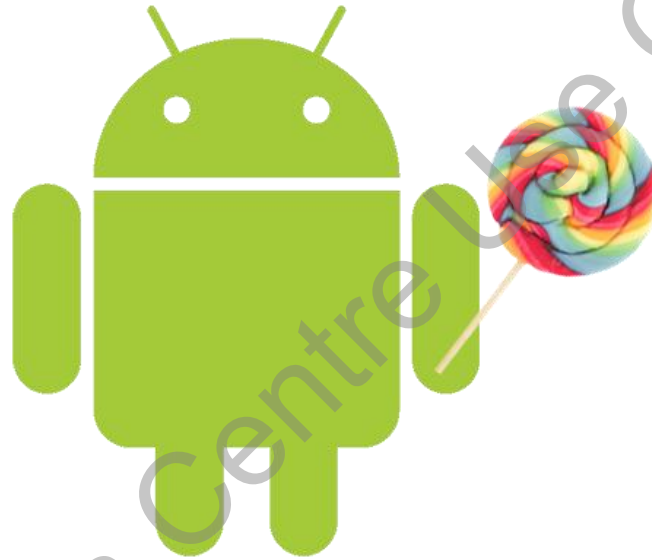


Programming in Android



Session: 12

Telephony, SMS, and VoIP

Objectives

- ◆ Explain Telephony
- ◆ Explain Telephony Manager
- ◆ Explain SMS
- ◆ Explain the process of sending SMS messages programmatically from within your application
- ◆ Explain the process of receiving incoming SMS messages
- ◆ Explain SIP and VoIP
- ◆ Use the SIP Framework to make a VoIP application

Introduction

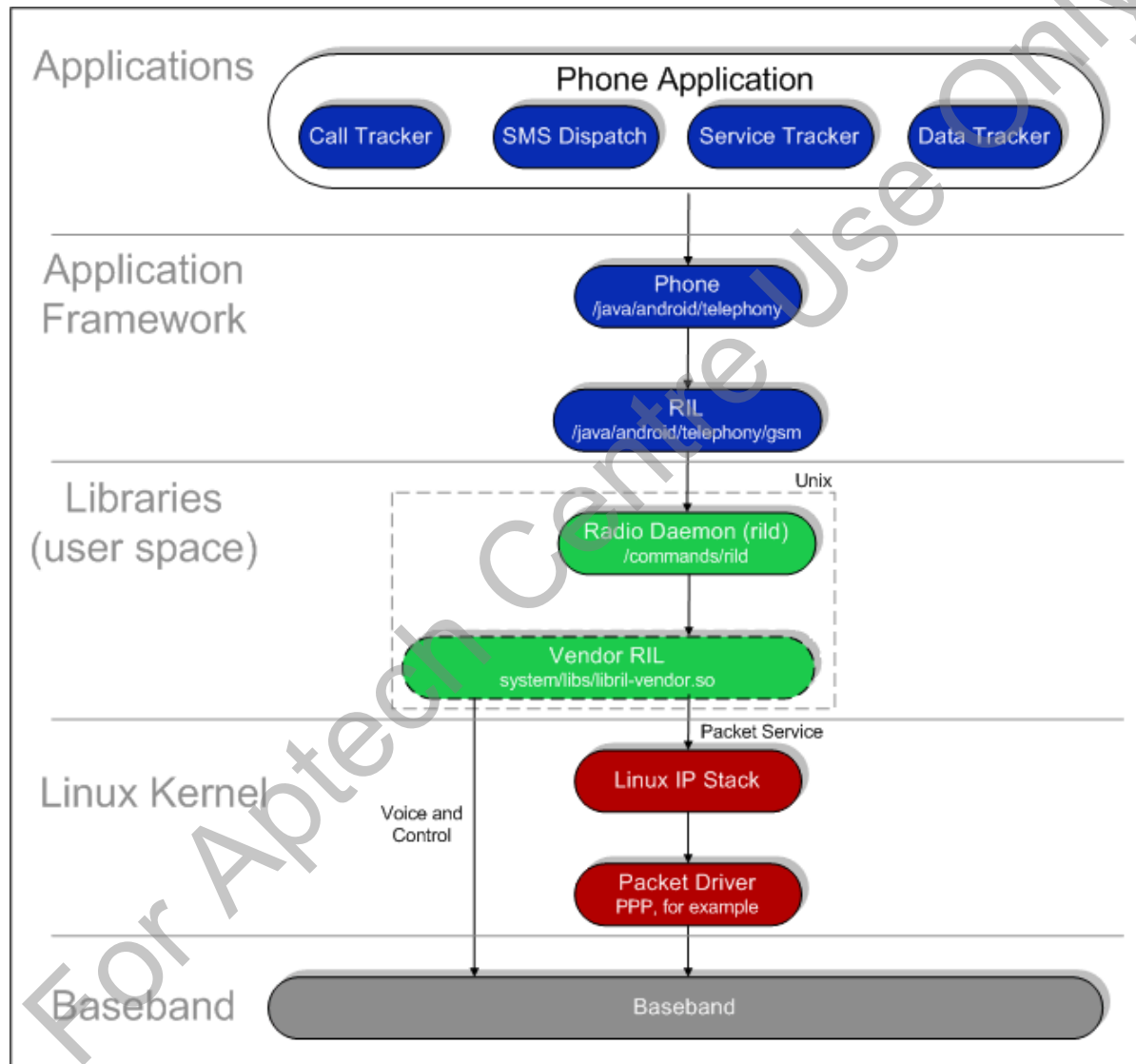
- ◆ Communication is essential to interactive applications
- ◆ Android provides easy-to-use API for SMS, SIP, and Telephony
- ◆ The RIL is the core component of the Telephony framework
- ◆ SIP is the standard framework for developing VoIP Applications



- ◆ Android devices supporting telephony services have a modem installed within them either as a separate module or as part of System on Chip (SoC)
- ◆ Android always views the modem as an autonomous device
- ◆ The communication with the modem is done with a serial port or an emulated serial port using Hayes AT Command Set
- ◆ Voice implementation varies from one modem to another



Telephony Framework 1-2

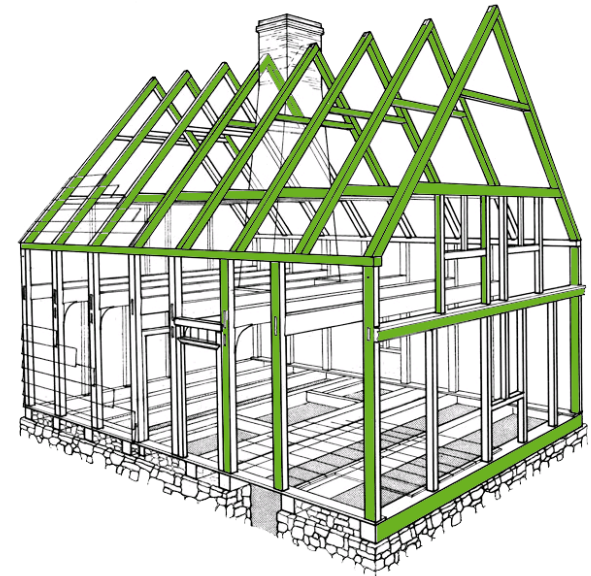


Telephony Framework 2-2

- ◆ The telephony framework provides services to the telephony application similar to the services provided by the Wi-Fi Manager
- ◆ This is the API that is exposed to the developer for creating applications
- ◆ The API is same across all the devices irrespective of the modem type or the AT commands supported by the device



- ◆ The telephony manager uses the API provided by the RIL Layer
- ◆ The RIL layer is where the AT commands and the modem interaction is executed
- ◆ It consists of the RIL Daemon which listens for requests and the RIL library which has the implementation to handle the requests
- ◆ The RIL implementation varies from one device to another
- ◆ The RIL layer translates the application requests into AT commands understandable by the modem



Linux Kernel Stack

- ◆ The Kernel stack is responsible for the drivers which support the modem device
- ◆ It is also responsible to provide a channel of communication for the RIL layer to send the AT commands
- ◆ In modern devices with SoCs, the Kernel stack is also responsible for initializing the modem hardware on startup



Flow of Commands

- ◆ A request is initiated from the caller application or the SMS application or a custom application using the telephony manager
- ◆ The request is received by the telephony framework. The telephony framework then creates a request for the RIL layer using a JNI interface
- ◆ RIL layer resolves the type of the request and executes the necessary Hayes commands
- ◆ The commands are sent over a serial port or an emulated port
- ◆ The Kernel layer translates the data from the emulated channel into the mode of communication used by the Modem (such as USB)

- ◆ Applications can use the methods present in this class to determine the telephony services and states, as well as have access to some types of subscriber information
- ◆ Applications can also register to a listener to receive notification of telephony state changes
- ◆ Apart from this, the telephony API is also used for activities such as monitoring phone information including the current states of the phone, connections, and network



Telephony Manager Example 1-2

- ◆ It provides information about the available telephony services and states on the device
- ◆ The application should have the required permission specified in the `AndroidManifest.xml`
- ◆ The code to create a Phone State Listener is shown in the following Code Snippet:

```
...
TelephonyManager telephonyManager;
...
telephonyManager = (TelephonyManager) getSystemService(Context.TELEPHONY_SERVICE);

phoneStateListener = new PhoneStateListener() {
    @Override
    public void onCallStateChanged(int state, String
        incomingNumber) {
    }
}
telephonyManager.listen(phoneStateListener, PhoneStateListener.LISTEN_CALL_STATE);
```

Telephony Manager Example 2-2

- Using the code, an application for demonstrating Telephony Manager functionality is created as shown in the following figure:



- ◆ SMS stands for Short Messaging Service
- ◆ SMS is one of the most important and frequently used applications that are executed on mobile devices
- ◆ Android comes with the built-in SMS application that enables you to send and receive SMS messages
- ◆ The SMSManager manages the SMS operations, such as sending text and data messages



Sending SMS Example 1-2

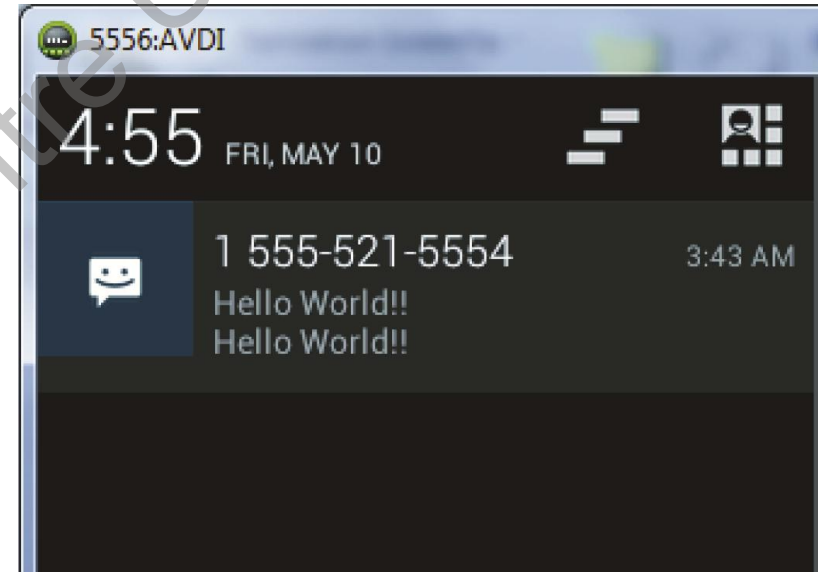
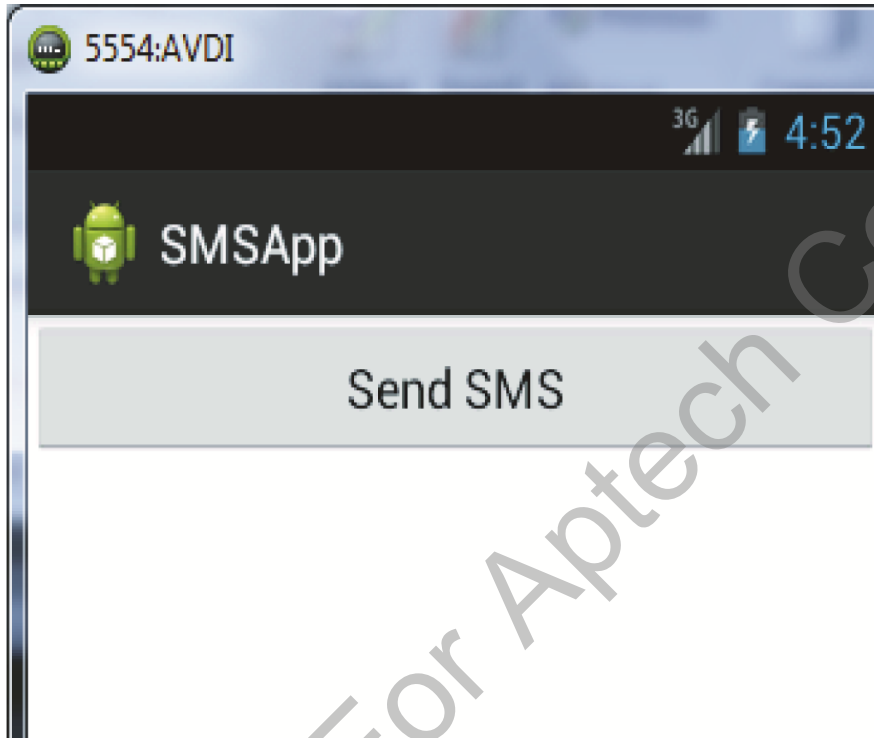
- ◆ The application can send SMS to another phone, when an event takes place
- ◆ In an Android application, the developer can access the SMS application using the SmsManager
- ◆ The `sendTextMessage()` function is used to send an SMS message as shown in the following Code Snippet:

```
...
SmsManager smsMgr = SmsManager.getDefault();
smsMgr.sendTextMessage("phone_num", null, "msg_text", null, null);
...

/*
public void sendTextMessage (String destinationAddress, String scAddress, String
text, PendingIntent sentIntent, PendingIntent deliveryIntent);
*/
```

Sending SMS Example 2-2

- Using the code, an application for demonstrating SMS functionality is created as shown in the following figure:
- When the SMS is sent and received by another device, the output will be as shown in the following figure:



Receiving SMS Example 1-2

- ◆ The developer can also receive incoming SMS messages from within the application using BroadcastReceiver object
- ◆ The code for this is shown in the following Code Snippet:

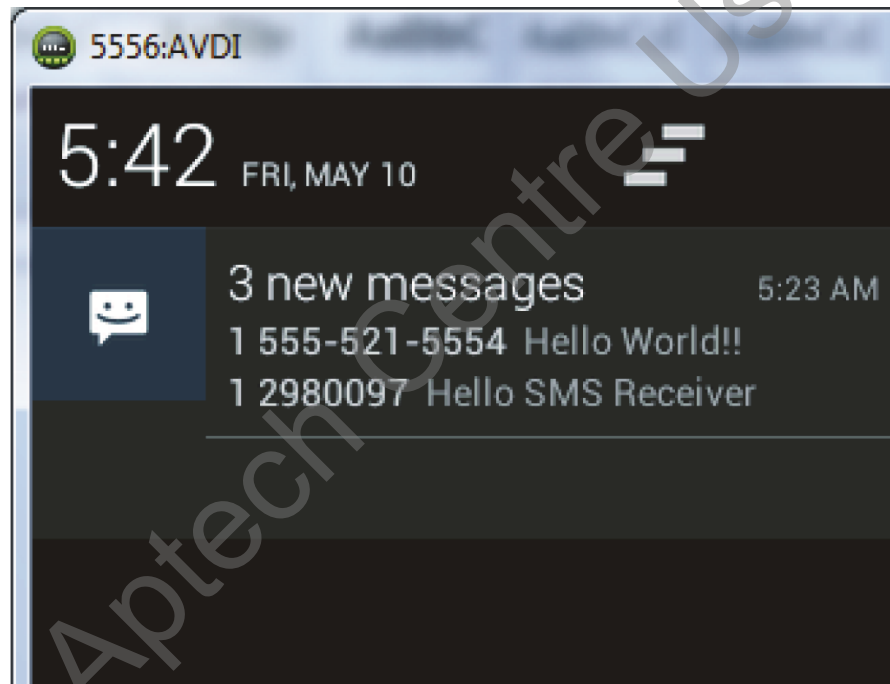
```
<receiver android:name=".SMSReceiver" >
<intent-filter>
<action android:name="android.provider.Telephony.SMS_RECEIVED" />
</intent-filter>
</receiver>
```

...

```
public class SMSReceiver extends BroadcastReceiver {
@Override
public void onReceive(Context , Intent intent) {
}
}
```


Receiving SMS Example 2-2

- Using the code, an application for demonstrating receiving SMS is created as shown in the following figure:



- The application creates a notification whenever an SMS is received

Sending SMS and Tracking it

- ◆ For tracking SMS, use broadcast receiver which checks the status of the message
- ◆ You have to develop your own BroadcastReceiver and pass the string using an object of the PendingIntent class
- ◆ Also, ContentObserver listening on the content://sms to track SMS
- ◆ This class will invoke the onChange() method when the SMS database changed



◆ VoIP

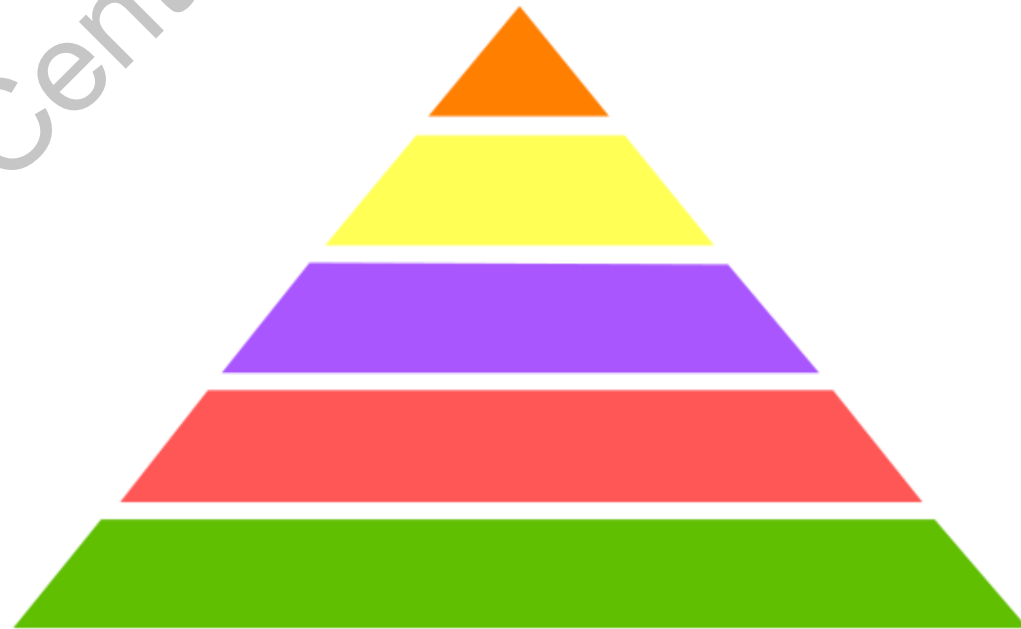
- ◆ VoIP is the abbreviation for Voice over Internet Protocol
- ◆ Transfer of voice over network has proven to be cheaper than the traditional telephone networks
- ◆ Most providers have a free service for VoIP as long as you have a data connection to their servers along with Mobile Clients
- ◆ These clients, in most cases, utilize the SIP protocol for signaling and data transfer

◆ SIP

- ◆ SIP is the abbreviation for Session Initiation Protocol
- ◆ SIP uses message based stateless protocol similar to HTTP and SMTP for session control
- ◆ Streaming protocols such as Real-time Streaming Protocol (RTP) and Secure Real-time Streaming Protocol (SRTP) are used for data transfer such as voice and video
- ◆ On a network level, SIP can utilize TCP, UDP, or SCTP



- ◆ The SIP model is similar to the SMTP mail transfer architecture
- ◆ It consist of:
 - ◆ SIP User Agent Client
 - ◆ SIP User Agent Server
 - ◆ Proxy Servers
 - ◆ Redirection Servers
 - ◆ Gateways
 - ◆ Registrars



SIP Requests 1-2

- SIP uses commands to generate requests
- SIP requests are listed in the following table:

Request	Description
INVITE	Indicates a client is being invited to participate in a call session
ACK	Confirms that the client has received a final response to an INVITE request
BYE	Terminates a call and can be sent by either the caller or the callee
CANCEL	Cancels any pending request
OPTIONS	Queries the capabilities of servers
REGISTER	Registers the address listed in the To header field with a SIP server
PRACK	Provisional acknowledgement

SIP Requests 2-2

Request	Description
SUBSCRIBE	Subscribes for an Event of Notification from the Notifier
NOTIFY	Notify the subscriber of a new Event
PUBLISH	Publishes an event to the Server
INFO	Sends mid-session information that does not modify the session state
REFER	Asks recipient to issue SIP request (call transfer)
MESSAGE	Transports instant messages using SIP
UPDATE	Modifies the state of a session without changing the state of the dialog

SIP Responses

- SIP uses numeric response codes as replies similar to the response codes used in HTTP
- The code families are in the following table:

Request	Description
1xx	Status Response codes
2xx	Success Response code
3xx	Redirection Response code
4xx	Client Side Error
5xx	Server Side Error
6xx	Protocol not supported
1xx	Status Response codes

◆ Relevant Classes

- ◆ **SipAudioCall**: This class is used to handle a voice call
- ◆ **SipSession**: This class is used to handle a SIP session
- ◆ **SipErrorCode**: This class is used to contain the error data
- ◆ **SipProfile**: This class holds connection information such as the server, user name, and password for the account
- ◆ **SipManager**: This class is responsible for establishing connection and provides access to SIP services

◆ Support Classes

- ◆ **SipAudioCall.Listener**: The listener class for and SipAudioCall. The methods for handling on incoming and on outgoing call events can be handled here. These events are also received by the SipSession.Listener class
- ◆ **SipSession.Listener**: The listener class for an SIP session
- ◆ **SipProfile.Builder**: The builder class for creating SipProfile objects
- ◆ **SipSession.State**: An encapsulation class holding integer values for call states

Permission and Perquisites

◆ Permissions

- ◆ In order to utilize SIP, the INTERNET and the USE_SIP permissions are mandatory
- ◆ Both these permissions need to be acquired by the application using the Manifest File

◆ Pre-requisites

- ◆ A device running Android Lollipop with a data connection. The SIP application needs to be deployed on a real device as AVD does not allow network connections
- ◆ A registered SIP account for use with the application
- ◆ Access to a SIP server

- ◆ Following Code Snippet demonstrates an example for logging into an SIP Server:

```
SipProfile.Builder builder = null;
try {
    builder = new SipProfile.Builder(userName, domain);
    builder.setPassword(passWord);
    builder.setAutoRegistration(true);
    profile = builder.build();

    // If AutoRegistration is set to false, uncomment the //below line of
code
    //sipManager.register(profile,9999,listener);

sipManager.open(profile,pendingIntent,listener);
```

SIP Incoming Call

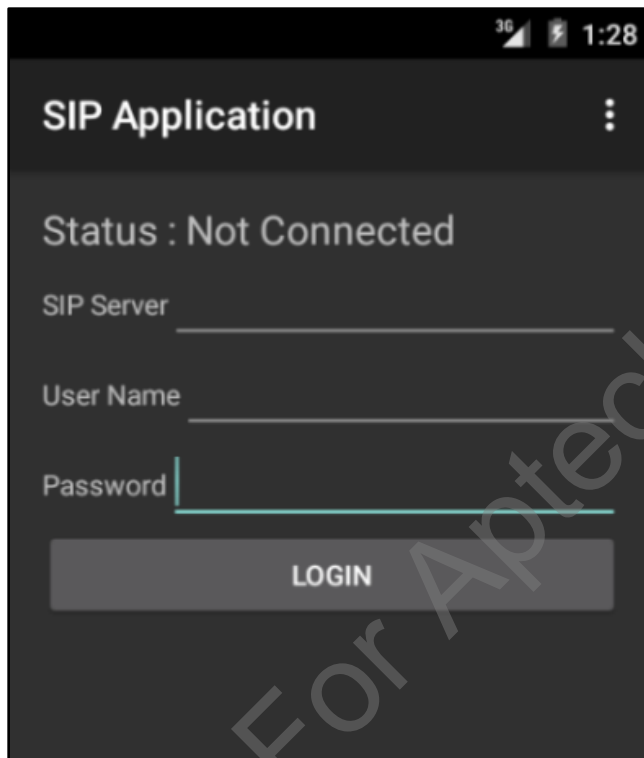
- ◆ Following Code Snippet demonstrates an example for receiving updates for incoming SIP calls:

```
receiver = new BroadcastReceiver() {  
    @Override  
    public void onReceive(Context , Intent intent) {  
        setStatus(" Incoming Call ");  
    }  
};  
  
filter = new IntentFilter();  
filter.addAction("sipApp.intent.IncomingCall");  
  
this.registerReceiver(receiver, filter);  
}
```

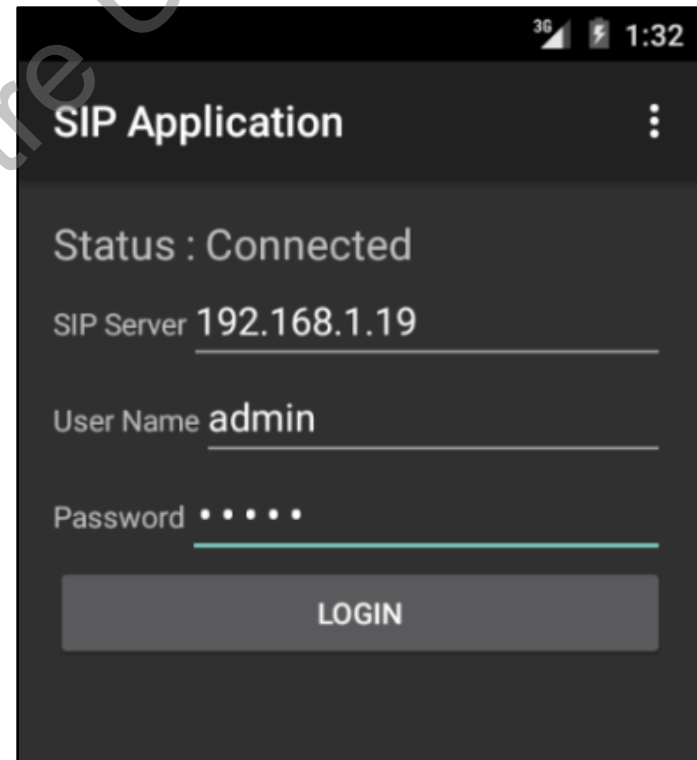
SIP Example Application 1-2

- Using the code, an application for demonstrating SIP framework is created as shown in the following figure:

- Once the connect button is clicked, the application registers with the SIP registrar the status is changed to connected as shown in the following figure:



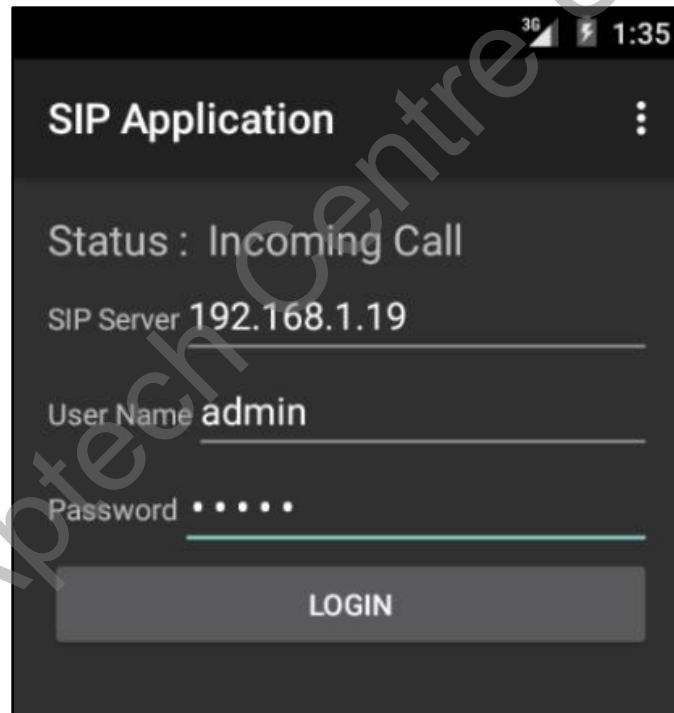
The screenshot shows a mobile application interface titled "SIP Application". The status is "Status : Not Connected". Below the status, there are three input fields: "SIP Server", "User Name", and "Password". A "LOGIN" button is at the bottom. The status bar at the top shows 3G signal, battery, and time 1:28.



The screenshot shows the same mobile application interface, but the status is now "Status : Connected". The "SIP Server" field contains "192.168.1.19", the "User Name" field contains "admin", and the "Password" field contains five dots. The "LOGIN" button is still present. The status bar at the top shows 3G signal, battery, and time 1:32.

SIP Example Application 2-2

- If the account receives an incoming call, the status is set to Incoming call as shown in the following figure:



Alternatives to SIP

- ◆ Alternatives to SIP protocols are available such as Extensible Messaging and Presence Protocol (XMPP) and Standard Interface for Multiple Platform Link Evaluation (SIMPLE)
- ◆ XMPP is an xml based instant messaging protocol that also supports voice and video data
- ◆ SIMPLE is an emerging standard closely based on the SIP standard
- ◆ Neither of these two protocols is natively supported by Android
- ◆ However, third party libraries are available to make the process of development much easier
- ◆ XMPP, in particular, has gained a huge market share

Summary

- ◆ Telephony provides access to information about the telephony services on the devices
- ◆ The Android TelephonyManager provides information about the Android telephony system. In other words, it provides information about the available telephony services and states on the device
- ◆ SMS stands for Short Messaging Service. SMS messaging is one of the most important and frequently used applications that is executed on the mobile devices
- ◆ The SmsManager manages the SMS operations, such as sending text and data messages. Similarly, ConnectivityManager manages the Internet connectivity in a mobile phone
- ◆ Apart from sending SMS messages from Android applications, the developer can also receive incoming SMS messages from within the application using BroadcastReceiver object
- ◆ VoIP is the process of utilizing network access to provide telephony services such as voice and video
- ◆ SIP is the protocol natively supported by Android for VoIP and similar applications