COMP-304
Fall 2018

Review of Lecture 9 – Networking API

- **Connect to web resources using HTTP:**
  - Use **URL** and **InputStream** classes
  - Use **HttpURLConnection** class to connect and get info about the resource
- Processing Asynchronously
  - Using **AsyncTask** class
  - Create a subclass of **AsyncTask**
    - **Override:**
      - onPreExecute()
      - doInBackground()
      - publishProgress()
      - onPostExecute()
- AsyncTask uses three parameters:
  - <TypeOfVarArgParams – passed to doInBackground
  - ProgressValue - progress information
  - ResultValue - returned from doInBackground()

Review of Lecture 9 – Networking API

- Call **execute** method on AsyncTask subclass constructor to run the task asynchronously.
- Accessing Web Services
  - Implement a method to call the web service and parse the XML
  - Use an XML parser
  - Create a subclass of AsynTask and call the web service asynchronously
- **Calling JSON services**
  - Implement a method to call the web service and parse JSON data
    - Use JSONArray and JSONObject objects
  - Create a subclass of AsynTask and call the web service asynchronously

SMS Messaging

Objectives:

- Students will learn:
  - How to send **SMS** messages **programmatically** from within your application
  - How to send **SMS** messages using the **built-in Messaging application**
  - How to **receive incoming SMS** messages
  - How to **send e-mail** messages from your application
  - How to use **WebView control** to browse the web, load content, etc.

Send SMS messages programmatically

- Add SMS permissions in the AndroidManifest .xml file

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

  - This permission enables users to decide whether to allow the application to install or not.

- To send an SMS message programmatically, you use the **SmsManager** class:

SmsManager sms = **SmsManager**.*getDefault();*

sms.**sendTextMessage**(phoneNumber, **null, message, null, null);**

Send SMS messages programmatically

- Following are the five arguments to the **sendTextMessage()** method:
  - **destinationAddress** — Phone number of the recipient
  - **scAddress** — Service center address; use null for default SMSC
  - **text** — Content of the SMS message
  - **sentIntent** — Pending intent to invoke when the message is sent
  - **deliveryIntent** — Pending intent to invoke when the message has been delivered

Getting Feedback after Sending a Message

- Create two **PendingIntent** objects to **monitor the status of the SMS message-sending process**.
  - Both **PendingIntent** objects are passed to the last two arguments of the **sendTextMessage()** method:

```
sentPI = PendingIntent.getBroadcast(this, 0,
new Intent(SENT), 0);
deliveredPI = PendingIntent.getBroadcast(this, 0,
new Intent(DELIVERED), 0);
```

  - *getBroadcast* - **Retrieves a PendingIntent that will perform a broadcast**
  - **PendingIntent** objects will be **used to send broadcasts later** when an SMS message has been **sent** (“SMS_SENT”) and **delivered** (“SMS_DELIVERED”).

Getting Feedback after Sending a Message

- **Create** and **register** two **BroadcastReceiver**s in the **onResume()** method:

  registerReceiver(smsDeliveredReceiver, **new IntentFilter(DELIVERED));**

  registerReceiver(smsSentReceiver, **new IntentFilter(SENT));**

- Both BroadcastReceivers listen for intents that match "SMS_SENT" and "SMS_DELIVERED"
- The two PendingIntent objects are passed into the last two arguments of the **sendTextMessage()** method:

  SmsManager sms = SmsManager.*getDefault();*

  sms.**sendTextMessage**(phoneNumber, **null, message, sentPI, deliveredPI);**

Getting Feedback after Sending a Message

- In the **onPause()** method, you **unregister** the two **BroadcastReceiver**s objects:

```
public void onPause() {
    super.onPause();
    //---unregister the two BroadcastReceivers---
    unregisterReceiver(smsSentReceiver);
    unregisterReceiver(smsDeliveredReceiver);
}
```

Sending SMS Messages Using Intent

- To activate the **built-in Messaging application** from within your application, you can use an Intent object together with the MIME type **"vnd.android-dir/mms-sms"**:

```
Intent i = new
Intent(android.content.Intent.ACTION_VIEW);
i.putExtra("address", "5556; 5558; 5560");
i.putExtra("sms_body", "Hello my friends!");
i.setType("vnd.android-dir/mms-sms");
startActivity(i);
```

Sending SMS Messages Using Intent

- You can send your SMS to multiple recipients by simply separating each phone number with a semi-colon (in the **putExtra()** method)
- Numbers will be separated using commas in the Messaging application.

Receiving SMS Messages

- Add the following permission:

<uses-permission
android:name=”*android.permission.RECEIVE_SMS*”/>

- To listen for incoming SMS messages, you create a **BroadcastReceiver** subclass
  - Override the **onReceive**(Context context, Intent intent) method
- When an incoming SMS message is received, the **onReceive()** method is fired.
- The SMS message is **contained in the Intent object** (intent; the second parameter in the onReceive() method) via a Bundle object.

Receiving SMS Messages

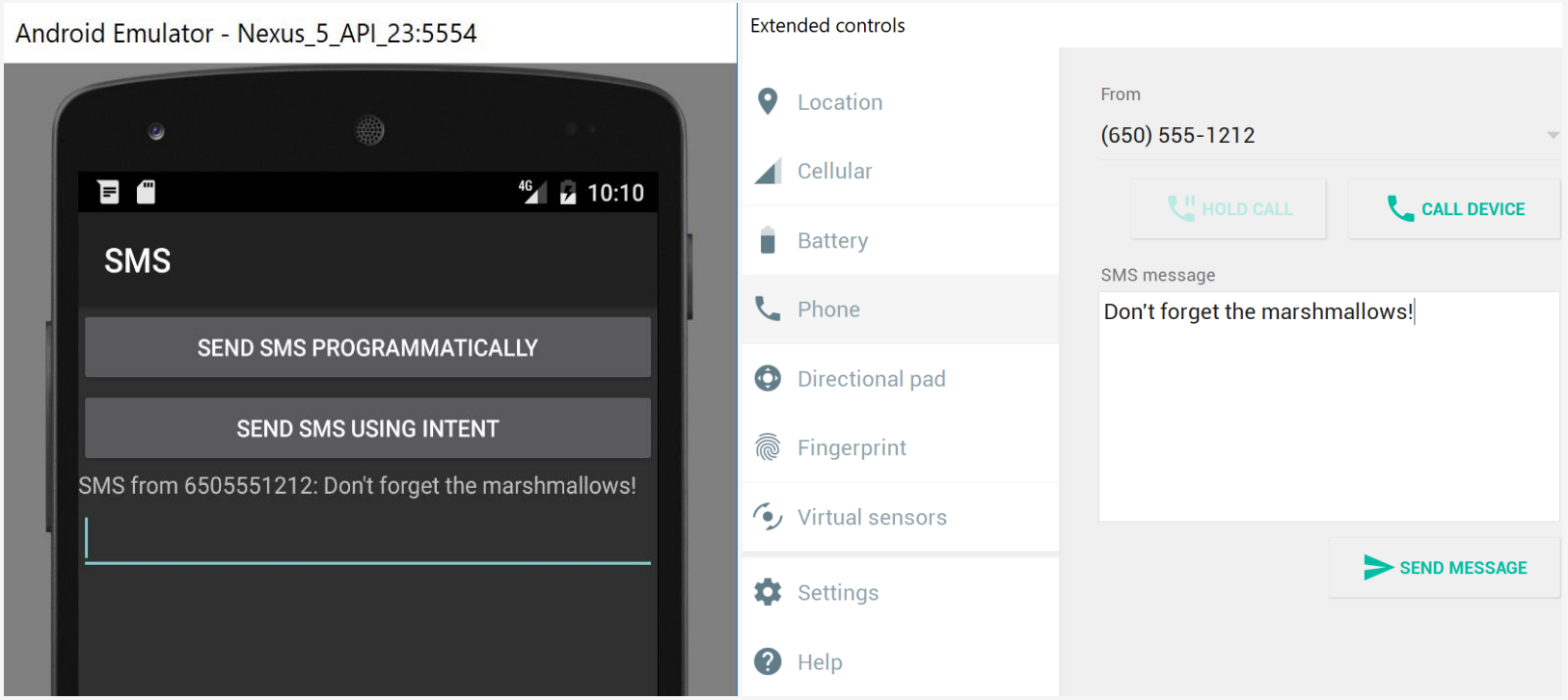
- Each SMS message is stored in an **Object** array in the PDU (**protocol data unit)** format.
  - If the SMS message is fewer than 160 characters, then the array will have one element.
  - If an SMS message contains more than 160 characters, then the **message will be split into multiple smaller messages** and stored as multiple elements in the array

Receiving SMS Messages

- To **extract the content of each message**, you use the static **createFromPdu()** method of **SmsMessage** class.
- The **phone number** of the sender is obtained via the **getOriginatingAddress()** method
- To **extract the body of the message**, you use the **getMessageBody()** method.
- Your **application will continue to listen for incoming SMS messages** even if it is not running; as long as the application is installed on the device, any incoming SMS messages will be received by the application.

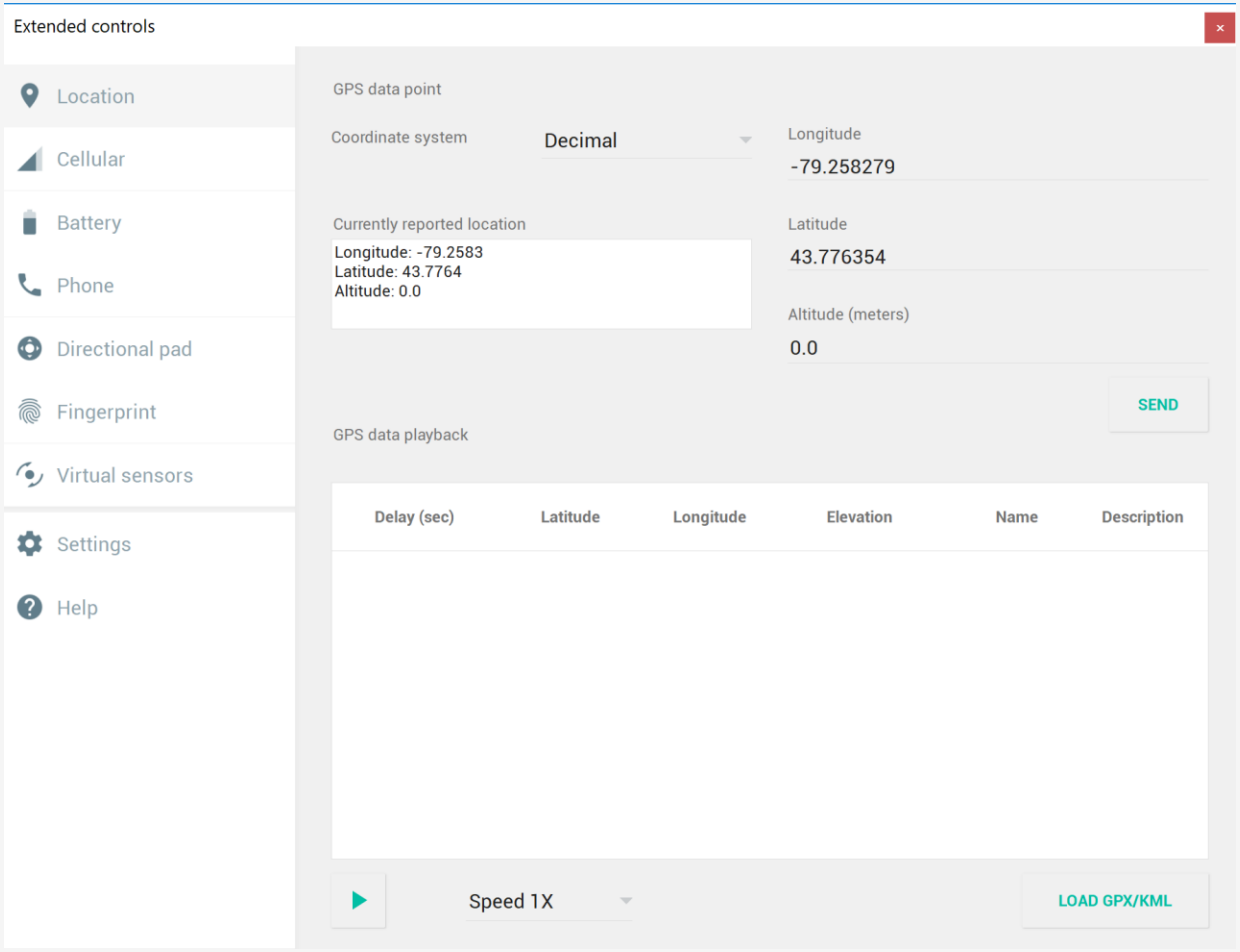
Receiving SMS Messages

- Example: Click more (…) on Emulator UI
- Select Phone
- Type the SMS message and click Send message:



Locating the Emulator

- Click more (…) on emulator, click Location, change lng, lat using Longitude, Latitude text boxes:



Preventing the built-in Messaging Application from Receiving a Message

- To prevent an incoming message from being handled by the built-in Messaging application, your application just needs to **handle the message before the built-in Messaging app** has the chance to do so.
- Add the **android:priority** attribute to the **<intent-filter>** element, like this:

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

Preventing the Messaging Application from Receiving a Message

- When an incoming message is received, your application will execute first
- To prevent other applications from seeing the message, simply call the **abortBroadcast()** method in your **BroadcastReceiver** class

Preventing the Messaging Application from Receiving a Message

```
@Override
public void onReceive(Context
    context, Intent intent)
{
//---get the SMS message passed in---
Bundle bundle = intent.getExtras();
SmsMessage[] msgs = null;
String str = "SMS from ";
if (bundle != null)
{
//---retrieve the SMS message received
Object[] pdus = (Object[])
    bundle.get("pdus");
msgs = new
    SmsMessage[pdus.length];
for (int i=0; i<msgs.length; i++){
msgs[i] =
    SmsMessage.createFromPdu((byte
    [])pdus[i]);
if (i==0) {
//---get the sender address/phone
number---
str += msgs[i].getOriginatingAddress();
str += ": ";
}
//---get the message body---
str +=
msgs[i].getMessageBody().toString();
}
//---display the new SMS message---
Toast.makeText(context, str,
Toast.LENGTH_SHORT).show();
Log.d("SMSReceiver", str);
//---stop the SMS message from being
broadcasted---
this.abortBroadcast();
}
}
```

Send the SMS message back to the main activity

In the **onReceive** method:

```
//---send a broadcast intent to update the SMS received
    in the activity---
Intent broadcastIntent = new Intent();
broadcastIntent.setAction("SMS_RECEIVED_ACTION
    ");
broadcastIntent.putExtra("sms", str);
context.sendBroadcast(broadcastIntent);
```

Send the SMS message back to the main activity

- In the main activity, get it in onReceive() method:

```
private BroadcastReceiver intentReceiver = new
    BroadcastReceiver() {
@Override
public void onReceive(Context context, Intent intent) {
//---display the SMS received in the TextView---
TextView SMSes = (TextView)
    findViewById(R.id.textView1);
SMSes.setText(intent.getExtras().getString("sms"));
}
};
```

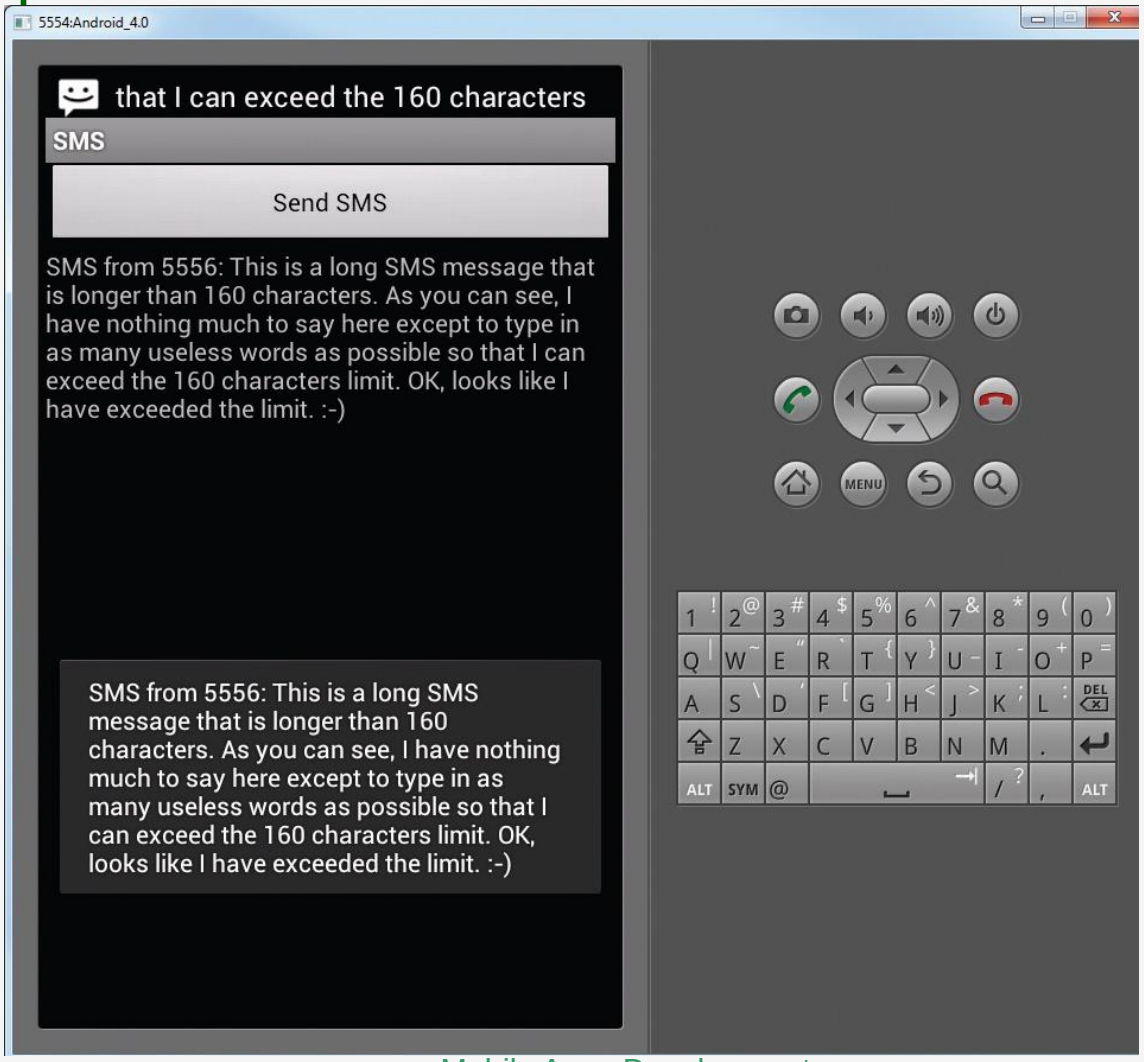
Send the SMS message back to the main activity

- In the **onCreate** method:

```
//---intent to filter for SMS messages received---
intentFilter = new IntentFilter();
intentFilter.addAction("SMS_RECEIVED_ACTION");
```

Send the SMS message back to the main activity

- Example:



SENDING E-MAIL

- Like SMS messaging, Android also supports e-mail.
- The Gmail/Email application on Android enables you to configure an e-mail account using POP3 or IMAP.
- You can also send e-mail messages programmatically from within your Android application.
- Invoke the **built-in Email application** to send an e-mail message.
- To do so, you use an **Intent** object and set the various parameters using the **setData()**, putExtra(), and **setType()** methods:

SENDING E-MAIL

```
//---sends an SMS message to another device---
private void sendEmail(String[] emailAddresses, String[]
    carbonCopies,
String subject, String message)
{
Intent emailIntent = new Intent(Intent.ACTION_SEND);
emailIntent.setData(Uri.parse("mailto:"));
String[] to = emailAddresses;
String[] cc = carbonCopies;
emailIntent.putExtra(Intent.EXTRA_EMAIL, to);
emailIntent.putExtra(Intent.EXTRA_CC, cc);
emailIntent.putExtra(Intent.EXTRA_SUBJECT, subject);
emailIntent.putExtra(Intent.EXTRA_TEXT, message);
emailIntent.setType("message/rfc822");
startActivity(Intent.createChooser(emailIntent, "Email"));
}
```

Browsing the Web with **WebView**

- Android applications can use the **WebView** control to **display web content** to the screen.
  - WebView control provides a **browser-like** view.
  - WebView control uses the **WebKit** rendering engine to draw HTML content on the screen.
- WebView control can display HTML pages on the Web or local files.
- WebKit is an **open source browser engine**
  - You can read more about it on its official website at http://**webkit.org**.

Browsing the Web with **WebView**

- Using the WebView control requires the **permission**.

- You should add *android.permission.**INTERNET*** permission to your application's Android manifest file as follows:

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

Browsing the Web with **WebView**

- **Launch the Browser application** using an Intent
  - When you want the user to have full access to all Browser features and having them return to your application when they're done:

```
Uri uriUrl = Uri.parse("http://androidbook.blogspot.com/");
Intent launchBrowser = new Intent(Intent.ACTION_VIEW, uriUrl);
startActivity(launchBrowser);
```

Designing a Layout with a **WebView Control**

- The WebView control can be added to a layout resource file like any other view.
- It can take up the entire screen or just a portion of it. A typical WebView definition in a layout resource might look like this:

```
<WebView
android:id=”@+id/web_holder”
android:layout_height=”wrap_content”
android:layout_width=”fill_parent”
/>
```

Loading Content into a **WebView Control**

- You can use **loadUrl** method to load content into a WebView control
- Here is how to use a WebView control to load content from a specific website:

```
final WebView wv = (WebView)
    findViewById(R.id.web_holder);
wv.loadUrl("http://www.perlgurl.org/");
```

- You can load an HTML file called *webby.html* stored in the application's **assets directory** like this:

```
wv.loadUrl("file:///android_asset/webby.html");
```

Parsing XML from the Network

- To render raw HTML, you can use the **loadData()** method:

String strPageTitle = “The Last Words of Oscar Wilde”;

String strPageContent = “<h1>” + strPageTitle +

“: </h1>\”Either that wallpaper goes, or I do.\””;

String myHTML = “<html><title>” + strPageTitle

+”</title><body>”+ strPageContent

+”</body></html>”;

wv.**loadData**(myHTML, “text/html”, “utf-8”);

Loading Content into a **WebView Control**

- Not all websites are designed for mobile devices.
- Change the scale of the web content to fit the page comfortably within the WebView control.
  - The call to the **setInitialScale()** method scales the view to 30 percent of the original size:

```
wv.setInitialScale(30);
```

- For pages that specify absolute sizes, scaling the view is necessary to see the entire page on the screen.
- Test and make page design changes (if the web content is under your control) for a good user experience.

Adding Features to the **WebView Control**

- The **WebView** control does not have all the features of a full browser.
  - it does not display the **title** of a webpage or provide buttons for reloading pages.
  - if the user clicks on a link within the WebView control, that action **does not load the new page within the view**
    - Instead, it **fires up the Browser application**.
- By default, all the WebView control does is display the web content provided by the developer using its internal rendering engine,WebKit.
- You can use **three classes**, in particular, to help modify the behavior of the control: the **WebSettings** class, the **WebViewClient** class, and the **WebChromeClient** class.

Modifying WebView Settings with WebSettings

- By default, a WebView control has various default settings: no zoom controls, JavaScript disabled, default font sizes, user-agent string, etc.
- You can change the settings of a WebView control using the **WebSettings** object.

```
final WebView wv = (WebView) findViewById(R.id.html_viewer);
WebSettings settings = wv.getSettings();
```

- The **getSettings()** method returns a **WebSettings** object that can be used to configure the desired WebView settings.

Modifying **WebView Settings with WebSettings**

- Some useful settings include:
  - Enabling and disabling zoom controls using the **setSupportZoom()** and **setBuiltInZoomControls()** methods
  - Enabling and disabling JavaScript using the **setJavaScriptEnabled()** method
  - Enabling and disabling mouseovers using the **setLightTouchEnabled()** method
  - **Configuring** font families, text sizes, and other display characteristics
- Here is an example:

```
settings.setJavaScriptEnabled(true);
```

- You can also use the WebSettings class to configure WebView plug-ins and allow for multiple windows.

Handling WebView Events with **WebViewClient**

- The **WebViewClient** class enables the application to listen for certain WebView events, such as when a **page is loading**, when a **form is submitted**, and when **a new URL** is about to be **loaded**.
- You can also use the **WebViewClient** class to determine and **handle any errors** that occur with page loading.
- You can tie a valid WebViewClient object to a WebView using the **setWebViewClient()** method.
- The following is an example of how to use **WebViewClient** to handle the **onPageFinished()** method to draw the title of the page on the screen:

Handling **WebView Events with WebViewClient**

```
WebViewClient webClient = new
    WebViewClient()
 { // anonymous implementation
      public void
          onPageFinished(WebView view,
          String url)
      {
            super.onPageFinished(view, url);
            String title = wv.getTitle();
            pageTitle.setText(title);
      }
};
wv.setWebViewClient(webClient);
```

Handling **WebView Events with WebViewClient**

- Sometimes when you load a page that redirects you to another page, WebView will cause your application to launch the device's Browser application to load the desired page
- To prevent this from happening, you need to implement the WebViewClient class and override the **shouldOverrideUrlLoading()**

Handling **WebView Events with WebViewClient**

```
public class WebViewActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        WebView wv = (WebView) findViewById(R.id.webview1);
        WebSettings webSettings = wv.getSettings();
        webSettings.setBuiltInZoomControls(true);
        wv.setWebViewClient(new Callback());
        wv.loadUrl("http://www.wrox.com");
    }
    private class Callback extends WebViewClient {
        @Override
        public boolean shouldOverrideUrlLoading(
            WebView view, String url) {
            return false;
        }
    }
}
```

Adding Browser Chrome with **WebChromeClient**

- You can use the WebChromeClient class in a similar way to the WebViewClient.
- However, WebChromeClient is specialized for the sorts of items that will be drawn outside the region in which the web content is drawn, typically known as *browser chrome.*
- *The* WebChromeClient class also includes callbacks for certain JavaScript calls, such as **onJsBeforeUnload()**, to confirm navigation away from a page.
- A valid WebChromeClient object can be tied to a WebView using the **setWebChromeClient()** method.

Adding Browser Chrome with **WebChromeClient**

- The following code demonstrates using **WebView** features to enable interactivity with the user.
- An **EditText** and a **Button** control are added below the **WebView** control, and a **Button** handler is implemented as follows:

```
Button go = (Button)
    findViewById(R.id.go_button);
go.setOnClickListener(new
    View.OnClickListener() {
public void onClick(View v) {
wv.loadUrl(et.getText().toString());
}
});
```

Adding Browser Chrome with **WebChromeClient**

- Using **WebChromeClient** can help add some typical chrome on the screen.
- You can use it to listen for changes to the title of the page, various JavaScript dialogs and console messages.

```
WebChromeClient webChrome = new WebChromeClient() {
        @Override
        public void onReceivedTitle (WebView view, String title)
        {
                Log.v(DEBUG_TAG, “Got new title”);
                super.onReceivedTitle(view, title);
                pageTitle.setText(title);
        }
};
wv.setWebChromeClient(webChrome);
```

References

- Textbook
- Android Documentation: https://developer.android.com/reference/android/telephony/SmsManager.html
- Joseph Annuzzi Jr., Lauren Darcey, Shane Conder: Introduction to Android Application Development: Android Essentials (5th Edition)