Lesson 8

Using WebViews

Victor Matos
Cleveland State University

WebViews

Android's Design Strategies

The Android platform offers three basic design patterns:

1. **Pure Android**
   Create a **native Android** app using the SDK and keep the app in the device. Such an app uses Android primitive resources such as widgets, services, activities, fragments, content-providers, notifications, etc.)

2. **Pure External HTML**
   Create **a remote website** and allow Android devices to fetch the external web-pages using the device's browser.

3. **Mixed Mode**
   Create **an internal website** hosted in the device. Allow the local HTML-pages (making the app) to interact with local Android resources.

Android's Design Strategies

Each approach offers **advantages & disadvantages**. For instance,

- Option (1) is richer in GUI and logic controls but is limited to Android devices. Porting the app to other platforms (say Apple's IOs or Windows Mobile) requires full re-write of all its parts.

- Option (2) is the most portable version. Any device with Internet access can interact with the external remote site; however this model does not use any of the multiple Android's hardware and software resources (with the exception of its system browser).

- Option (3) is an interesting in-between compromise. Solutions based on this approach are –at least in principle- more 'easily' transferred from one platform to the other. Current improvements in the HTML5 standard make the strategy even more appealing.

WebViews

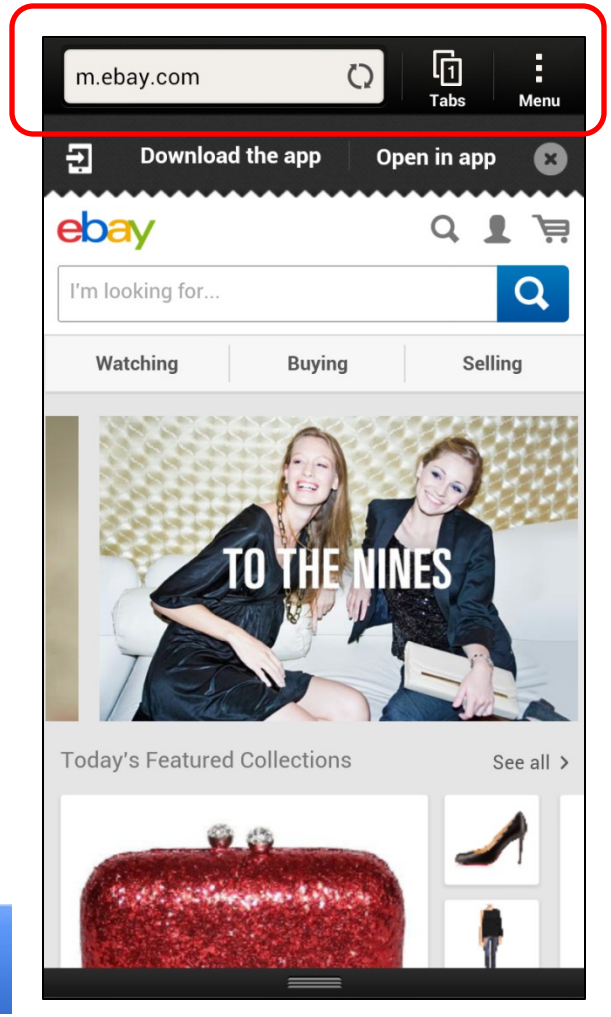
WebView Features

1. **WebViews** use the **WebKit** Open Source Engine (this engine is also present on other systems such as Apple's IOs - http://www.webkit.org)

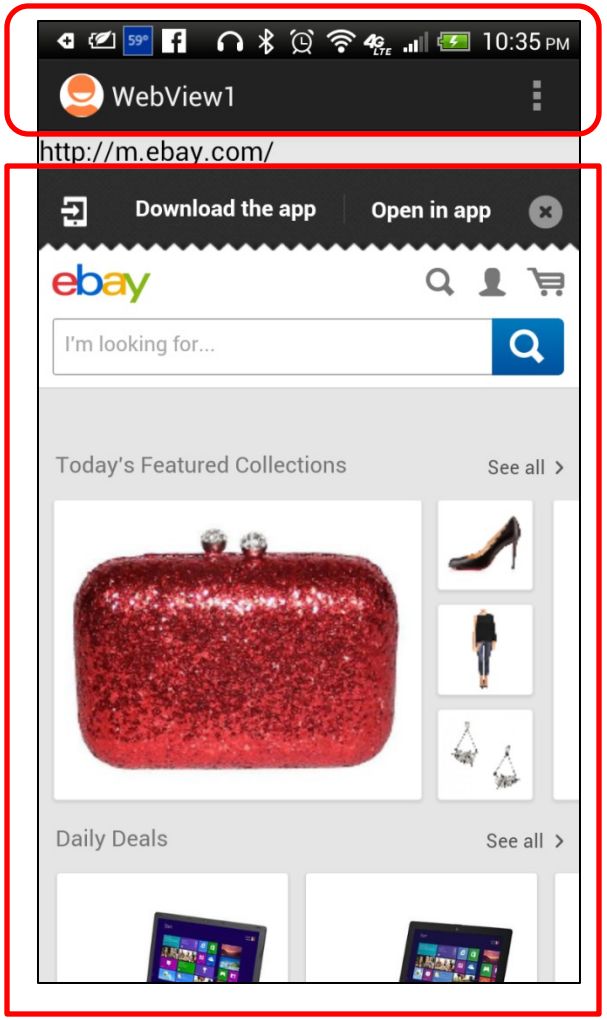
2. In principle, a WebView is not exactly a regular browser. Although it is used to show webpages, it does not include the common browser's buttons such as Back, Forward, Reload, etc.

3. The **WebView** class includes (among others) methods to:
   - Load /Reload a page, navigate forward and backward through a history record,
   - Zoom in and out,
   - Use CSS and JavaScript features to provide different styles and images based on the screen's pixel density.
   - Exchange data with the Android device using a JavaScript Interface
   - Perform text searches, capture pictures, load data / stop loading …
   - Implementing those methods and services could be done in various ways, for instance through Menus, ActionBar, Buttons, etc.

WebViews



Android's browser showing a webpage on an HTC phone. Originally www.ebay.com was requested, but redirected to http://m.ebay.com

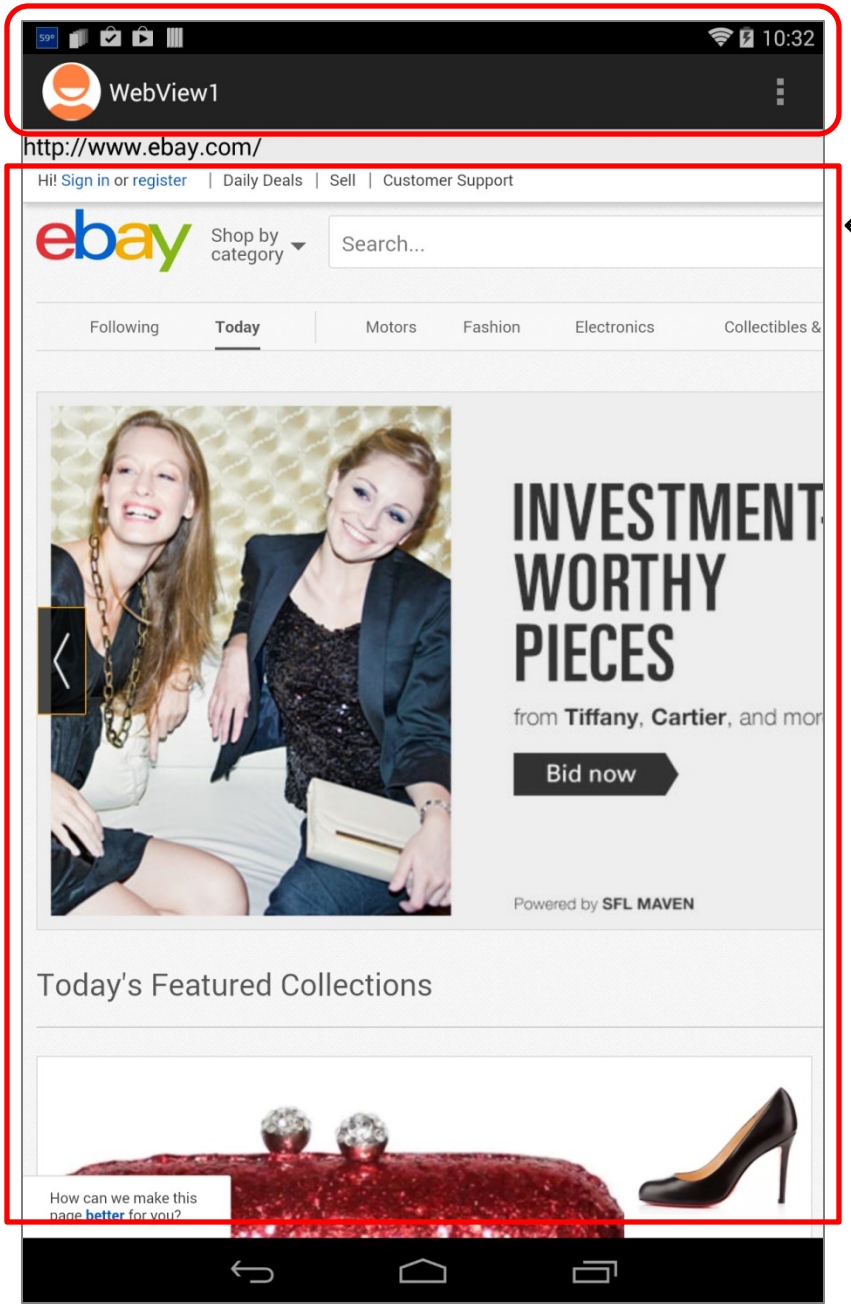


←**WebView** Showing redirected page at m.ebay.com

Android **app** (Example1A) using an embedded **WebView** to show the same webpage illustrated on the figure to the left. Image obtained from an HTC phone.

WebViews

Android **app** (Example1A) running on a tablet. The original requested site www.ebay.com is shown as it would be on a 'normal' browser running on a laptop/desktop computer (*no redirection*).



←**WebView**

Adding Permissions

Warning! In order for your Activity to access the Internet you must add the **INTERNET** permission to your Android **Manifest** file. For using add-on libraries such as **Google Maps**, you need an explicit <uses-library…> clause included in your Manifest.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="csu.matos.webview_demo1"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="18" />
    <uses-permission android:name="android.permission.INTERNET"/>

    <application
        . . .
        <activity
            . . .
        </activity>
        <uses-library android:name="com.google.android.maps" />
    </application>

</manifest>
```

WebViews

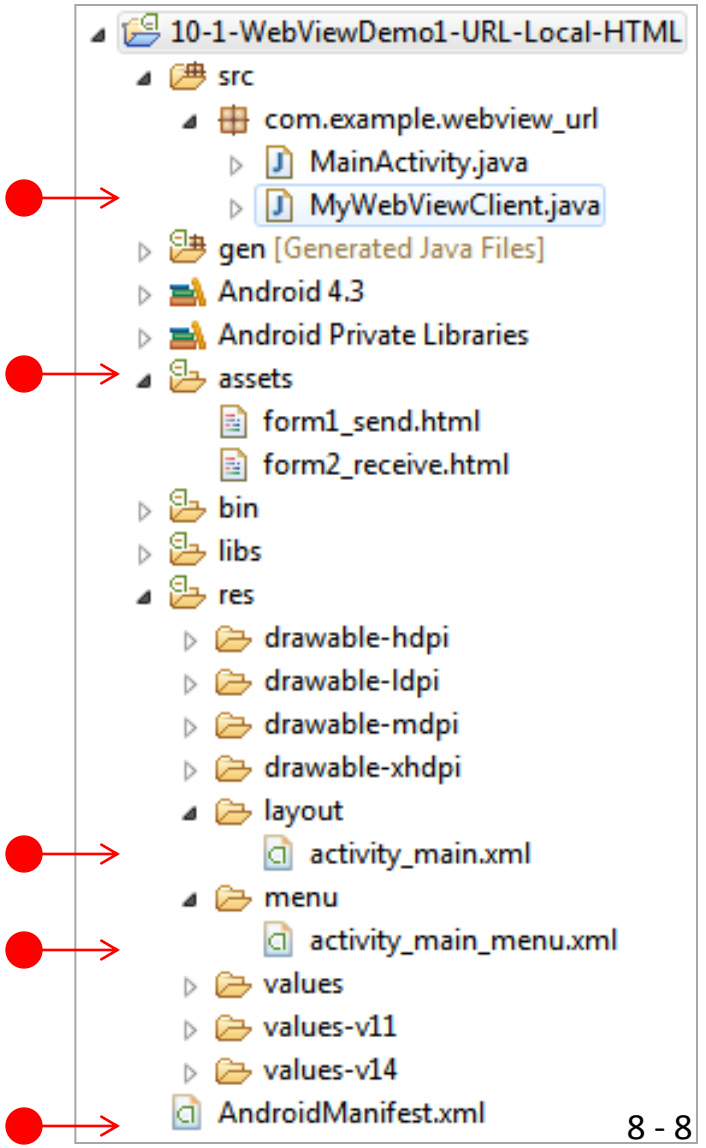
Example 1. Populating a WebView

In this example we explore three different ways in which a WebView could be populated:

1. First, our WebView is loaded with an external web-page whose URL is known to us.

2. A second case is shown in which a WebView exposes a set of locally stored HTML pages.

3. A third WebView is used to display an HTML page whose code is dynamically supplied by the app.

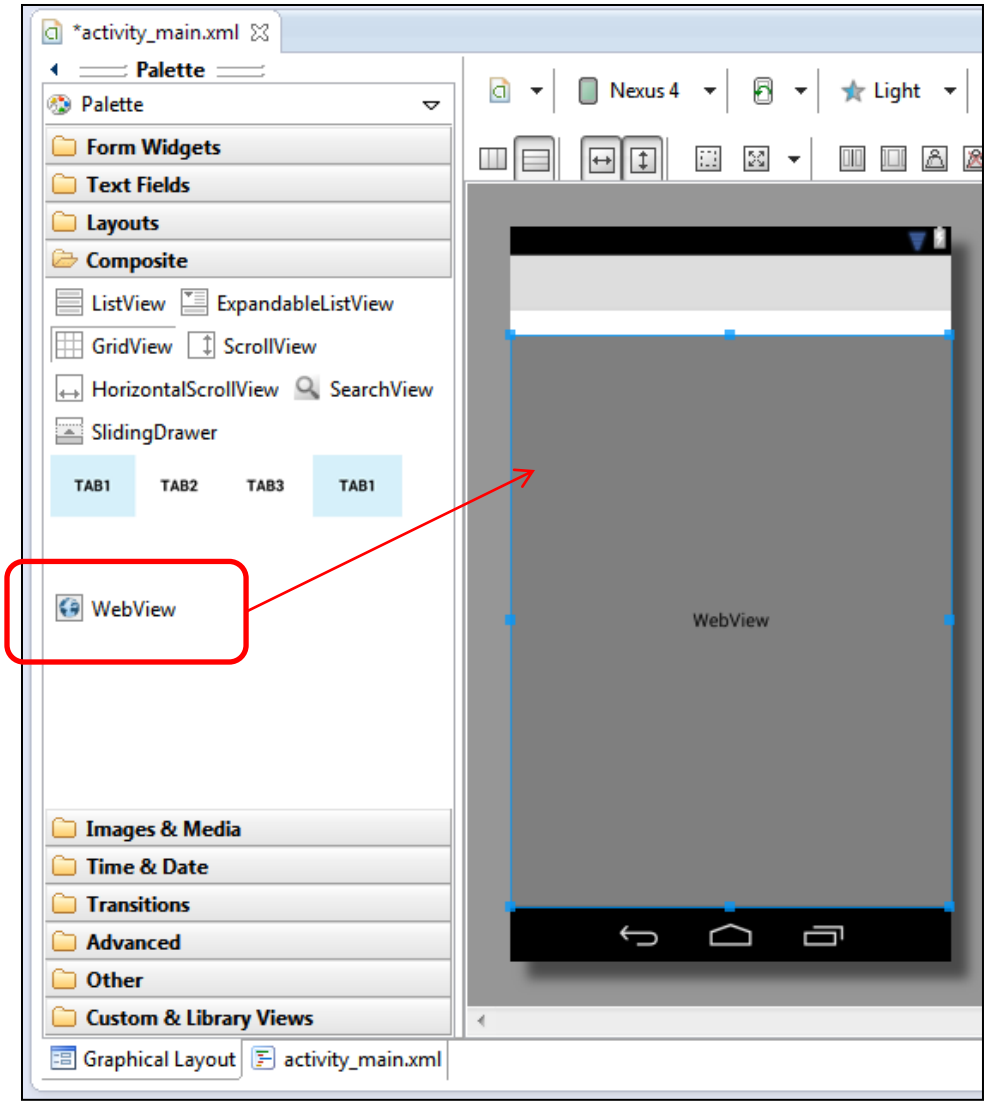


WebViews

Example 1. Populating a WebView

Constructing the example

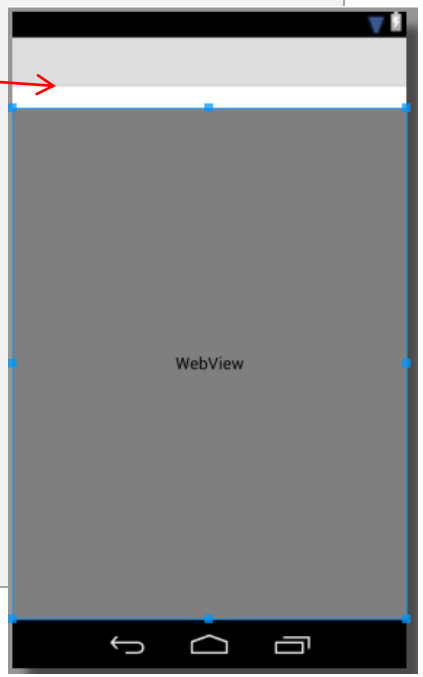
Eclipse users can find the **WebView** widget in the *Composite* pane of the GUI Editor.



WebViews

Example 1. Layout – activity_main.xml

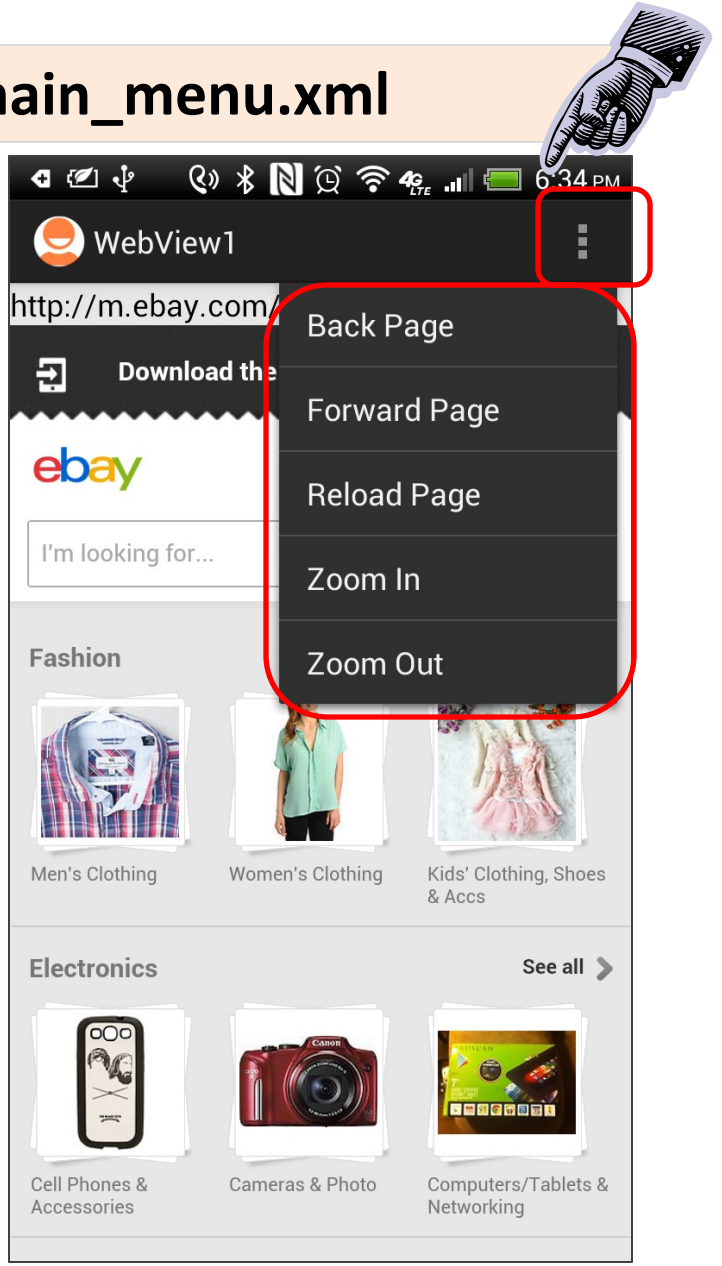
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"  >

    <TextView
        android:id="@+id/txtMsg"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <WebView
        android:id="@+id/webView1"
        android:layout_width="match_parent"
        android:layout_height="match_parent"  />

</LinearLayout>
```



WebViews

Example 1. Option Menu – activity_main_menu.xml

```
<menu
xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:id="@+id/back_page"
        android:orderInCategory="100"
        android:showAsAction="never"
        android:title="Back Page"/>
    <item
        android:id="@+id/forward_page"
        android:orderInCategory="110"
        android:showAsAction="never"
        android:title="Forward Page"/>
     <item
        android:id="@+id/reload_page"
        android:orderInCategory="120"
        android:showAsAction="never"
        android:title="Reload Page"/>
     <item
        android:id="@+id/zoom_in"
        android:orderInCategory="130"
        android:showAsAction="never"
        android:title="Zoom In"/>
     <item
        android:id="@+id/zoom_out"
        android:orderInCategory="140"
        android:showAsAction="never"
        android:title="Zoom Out"/>
</menu>
```



WebViews

Example 1. MainActivity.java

```
public class MainActivity extends Activity {

   TextView txtMsg;
   WebView webview;


   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      txtMsg = (TextView) findViewById(R.id.txtMsg);

      // Try demo1, demo2, or demo3 (please, uncomment one at the time!!!)
       demo1TrySpecificUrl();

      // demo2TryLocallyStoredHtmlPage();
      // demo3TryImmediateHtmlPage();
      // demo4TryRichGoogleMap();

   }// onCreate

   // ----------------------------------------
   // Demo1, Demo2 and Demo3 code goes here...
   // ----------------------------------------

}//MainActivity
```

WebViews

Example 1. MainActivity - (Demo1) Show a Remote Page

```
@SuppressLint("SetJavaScriptEnabled")
private void demo1TrySpecificUrl() {
   webview = (WebView) findViewById(R.id.webView1);
1  webview.getSettings().setJavaScriptEnabled(true);
   //webview.setWebViewClient(new WebViewClient());  //try later
   //set ebay.com as "home server" - go do some shopping
2  webview.setWebViewClient(new MyWebViewClient(txtMsg, "ebay.com"));
3  webview.loadUrl("http://www.ebay.com");
   //webview.loadUrl("http://www.amazon.com"); //try later

}
```

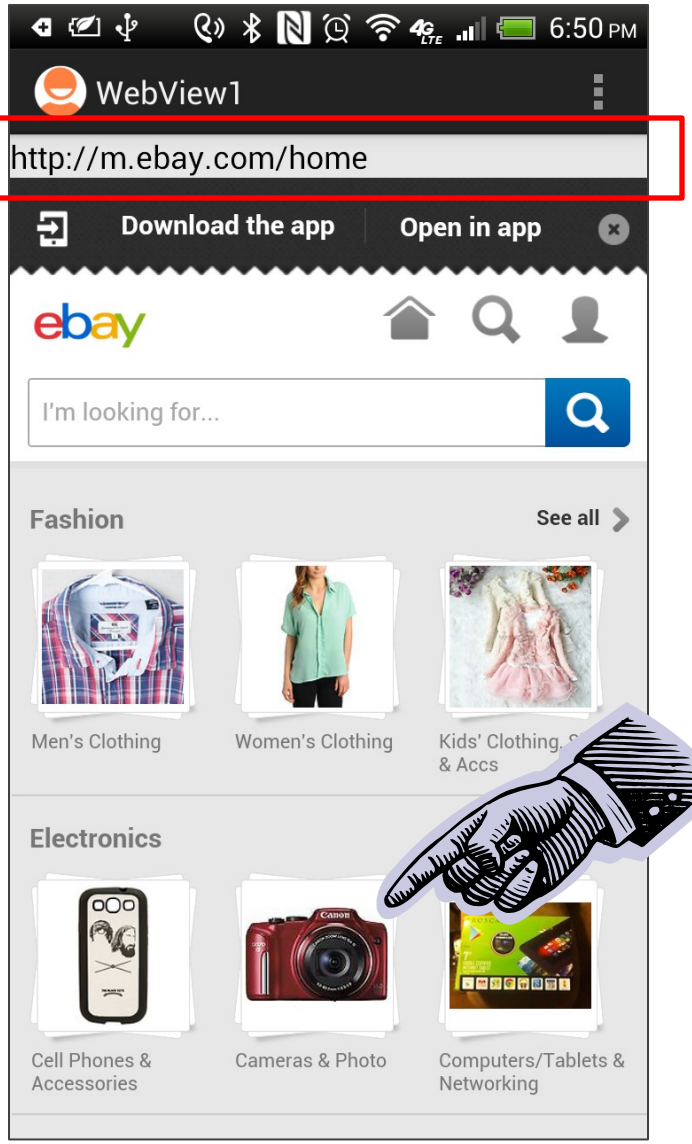
This app is hard-wired to **eBay.com** as "home server"

Comments

1. Allow the visited page to execute its **JavaScripts** functions (if any)
2. The extended class **MyWebViewClient** is asked to control page traffic and update the GUI showing the actual URL used by the browser. In our example, only pages from ebay.com are allowed to be shown on the WebView widget. Other sites will be displayed in a separate browser (*more on this class at the end of the example*)
3. The given **URL** is used to initiate the Internet navigation process. Caution, the final page could be different from the original address due to redirection enforced by the remote server. In this example we asked to visit www.ebay.com but ended up at m.ebay.com (when using small screens!).

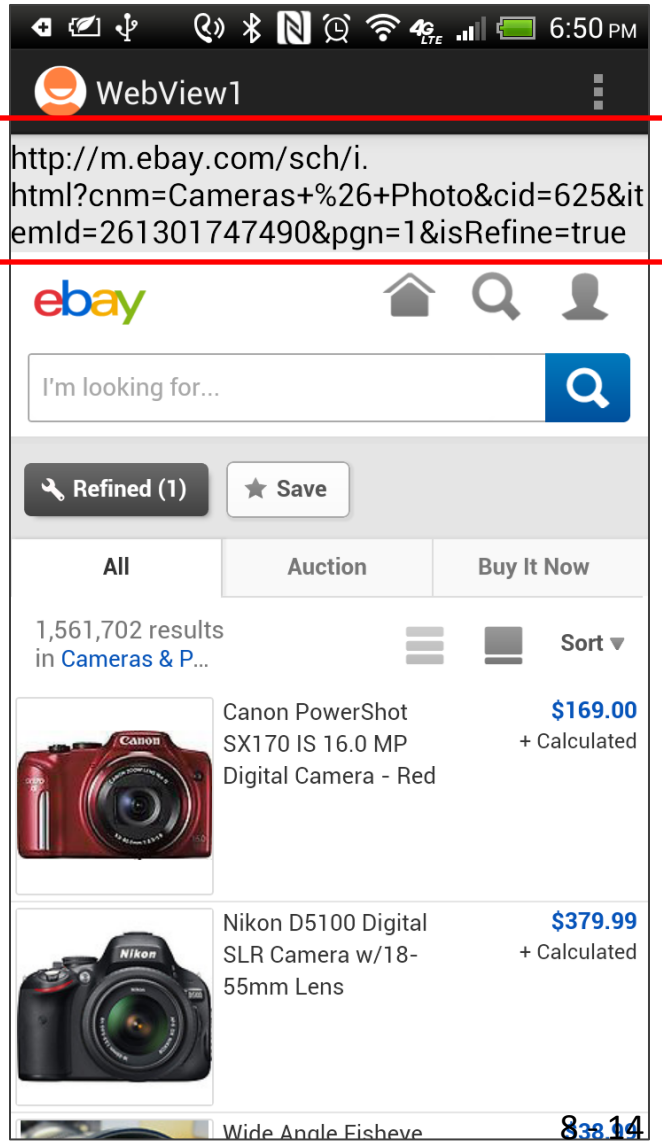
WebViews

Example 1. MainActivity - (Demo1) Show Remote Page



← Web Page To which the app is redirected

Click to navigate forward. Use Menu for other options.



WebViews

Example 1. MainActivity - (Demo2) Show Local Pages

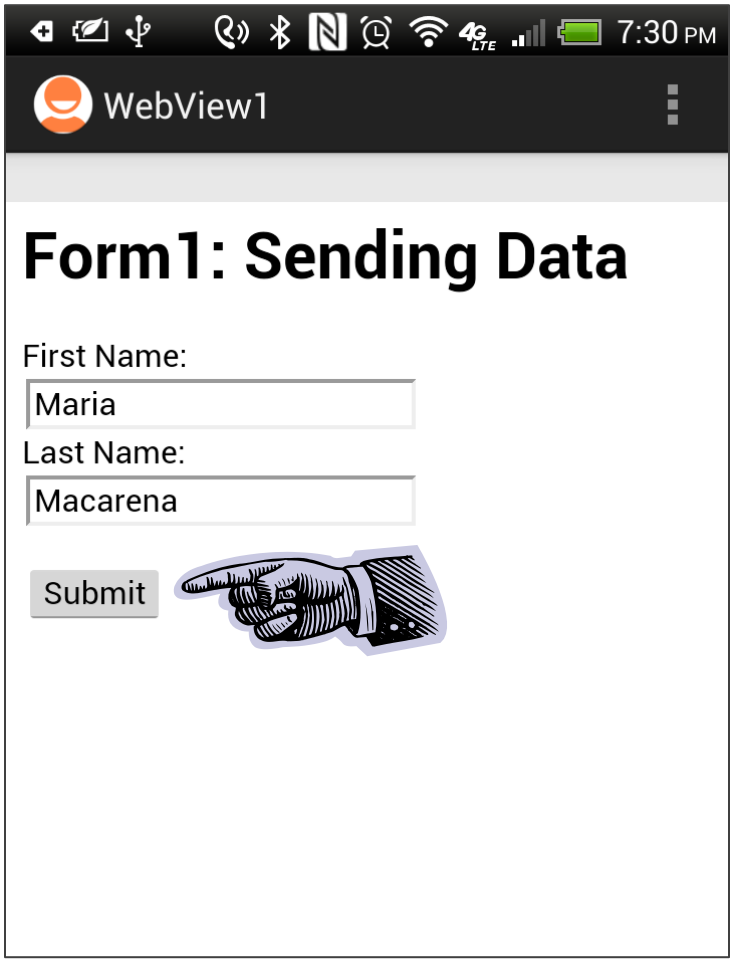
```
@SuppressLint("SetJavaScriptEnabled")
 private void demo2TryLocallyStoredHtmlPage() {

    webview = (WebView) findViewById(R.id.webView1);
    webview.getSettings().setJavaScriptEnabled(true);

    // a custom WebViewClient could check for url containing the
    // home-base address "file:///android_asset/"
1 → webview.setWebViewClient(new WebViewClient()); //continue using WebViews
    //webview.setWebViewClient(new MyWebViewClient2(this)); //try later
2 → webview.loadUrl("file:///android_asset/form1_send.html");

 }//demo2TryLocallyStoredHtmlPage
```

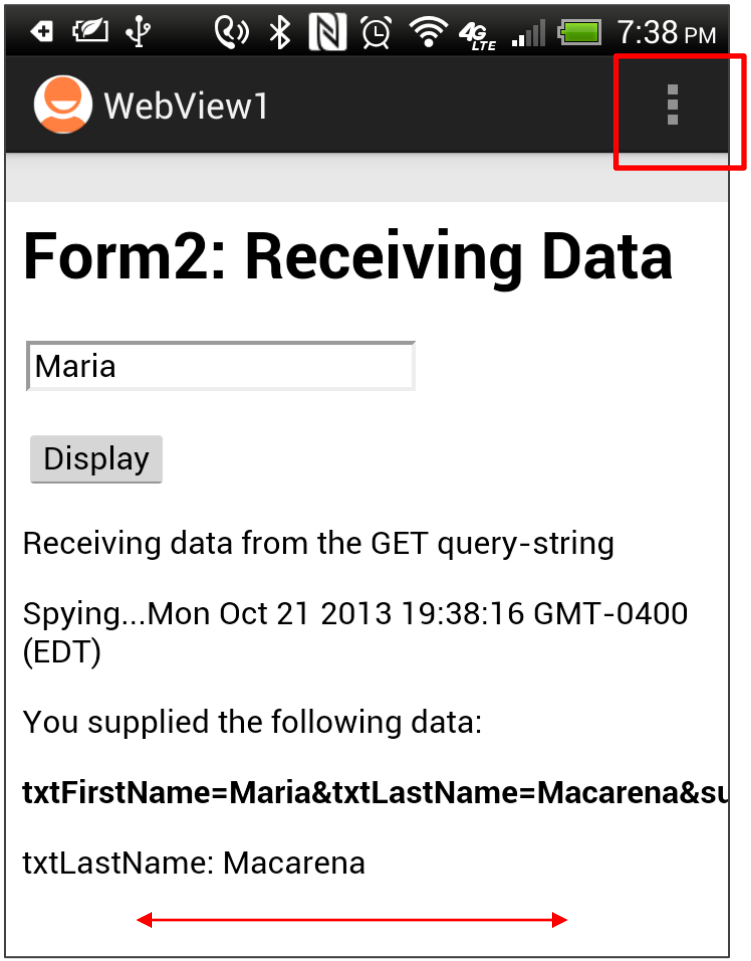
Comments

1. Using a *default* **WebViewClient** to control page traffic. By default requested URLs are all –without exception- shown in the WebView (no browsers called). A record of all visited links is kept so forward-backward navigation could be used. In this example the 'home-server' address begins with: `file:///android_asset/`
2. The web-page to be shown is locally stored in the app's memory space (it could also be on SDcard). In this example, our pages are held in the **res/asset/** folder.

WebViews

Example 1. MainActivity - (Demo2) Show Local Pages





Comment: Click on the **Submit** button to call the second web-page passing form-data (user's First & Last name)

WebViews

Example 1. MainActivity - (Demo2) Show Local Pages

```
<html>
<head>
</head>
<body>

<FORM action="form2_receive.html" id=form1 method=GET name=form1>
          <h1>Form1:  Sending Data</h1>
          First Name:
          <br>
          <INPUT id="txtFirstName" name="txtFirstName" value="Maria">
          <br>
          Last Name:
          <br>
          <INPUT id="txtLastName" name="txtLastName" value="Macarena">
          <p>
          <INPUT type="submit" value="Submit" id=submit1 name=submit1>
</FORM>
</body>
</html>
```

Comments

After pressing **Submit** the page *form2_receive.html* will be called. Current form data will be supplied using the **HTTP GET** method. Therefore the QueryString will include:

`txtFirstName=Maria&txtLastName=Macarena&submit1=Submit`

WebViews

Example 1. form2_receive.html - (Demo2) Show Local Pages

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
   <title>Android & Plain HTML Demo</title>
   <script language="javascript">

  function extracting(variable) {
    var query = window.location.search.substring(1);
    alert(query);
    var personName = getQueryVariable(variable)
    document.getElementById("myMsg").value = personName;
  }

  function getQueryVariable(variable) {
      var query = window.location.search.substring(1);
      var vars = query.split('&');
     for (var i=0;i<vars.length;i++) {
          var pair = vars[i].split("=");
          if ((pair[0] == variable) || (variable == 0)) {
            return pair[1];
          }
       }
       alert('Query Variable ' + variable + ' not found');
   }
   </script>
```

Example 1. form2_receive.html – (Demo2) Show Local Pages

```
</head>
 <body>
 <h1>Form2:  Receiving Data</h1>

 <p><input TYPE="text"  id="myMsg" ></p>

 <p><input TYPE="button" value="Display"
           onClick="extracting('txtFirstName');"></p>


   <p>Receiving data from the GET query-string
   <script>
      document.write("<p>Spying..." + new Date() );
      document.write("<p> You supplied the following data:");
      var querystring = window.location.search.substring(1);
      document.write("<p><b>" + querystring + "</b>");

      document.write("<p>txtLastName: "  + getQueryVariable('txtLastName') );

   </script>


 </body>
</html>
```

WebViews

Example 1. form2_receive.html – (Demo2) Show Local Pages

Comments

The Browser-Object-Model (**BOM**) class **Location** contains information about the current URL. In particular, the property **window.location.search** returns the query portion of a URL, including the question mark '**?**'. In our example

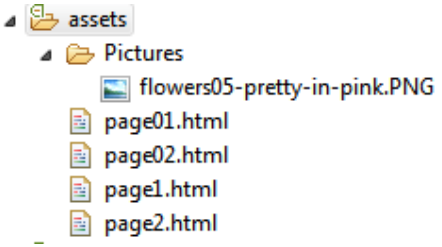
```
window.location.search is
?txtFirstName=Maria&txtLastName=Macarena&Submit1=Submit
```

Therefore

```
window.location.search.substring(0) is '?....' and

window.location.search.substring(1) is
txtFirstName=Maria&txtLastName=Macarena&submit1=Submit
```

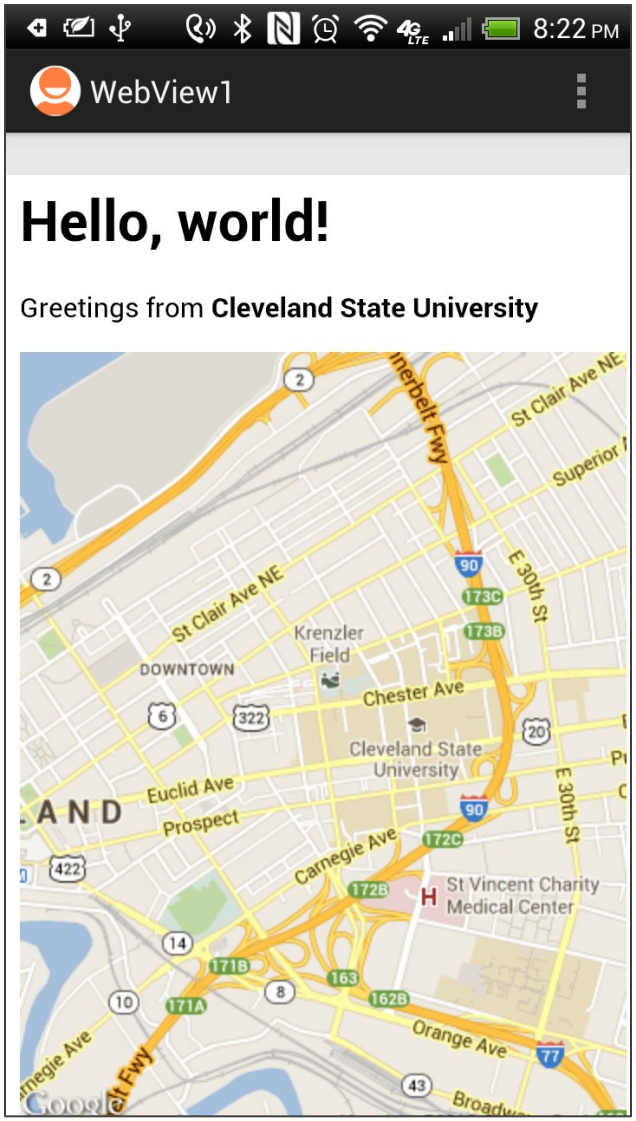
You may add *images* to Form1. Add sub-directory **/pictures** to the **assets** folder. Refer to a picture using a clause such as:

```
<img src="pictures/picture_name.png" />
```



WebViews

Example 1. MainActivity - (Demo3) Show Immediate Page



Playing with Google Maps V2 (Using Coordinates):
Use your computer to access the following link

http://maps.googleapis.com/maps/api/staticmap?center=41.5020952,-81.6789717&zoom=14&size=350x450&sensor=false

It displays a web page showing a static map centered around the given coordinates (latitude and longitude of Cleveland State University Student's center)

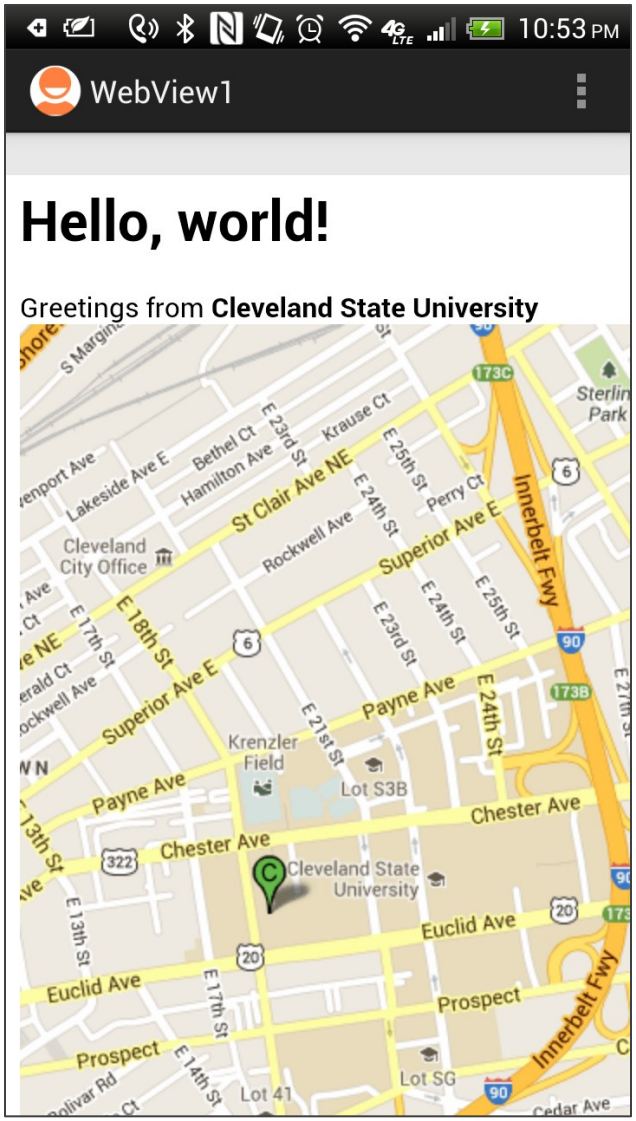
We want to reproduce this behavior in our Android app.

This image was generated using the **Google Maps Image API**. For more information visit https://developers.google.com/maps/

For converting street-coordinates see:
http://stevemorse.org/jcal/latlon.php

WebViews

Example 1. MainActivity - (Demo3) Show Immediate Page



Playing with Google Maps V2 (Mapping by Name):
Use your computer to access the following link

http://maps.googleapis.com/maps/api/staticmap
?center=Cleveland+State+University,Ohio
&zoom=15&size=480x700
&maptype=roadmap
&markers=color:green|label:C|41.5020952,-81.6785000
&sensor=false

The URL displays a web page showing a static roadmap of the given location. Reverse geo-location allows us to locate a place by it's name.

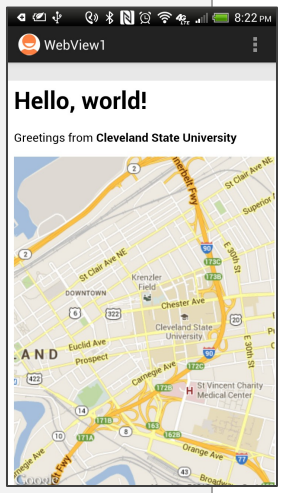
We want to reproduce this view in our Android app.

This image was generated using the **Google Maps API**. For more information visit https://developers.google.com/maps/

WebViews

Example 1. MainActivity - (Demo3) Show Immediate Page

```
@SuppressLint("SetJavaScriptEnabled")
private void demo3TryImmediateHtmlPage() {
    //make a WebView to show a Google Map for given coordinates
   webview = (WebView) findViewById(R.id.webView1);
   webview.getSettings().setJavaScriptEnabled(true);
   webview.setWebViewClient(new WebViewClient());

   String aGoogleMapImage = "<img src=\"http://maps.googleapis.com/maps/api/"
         + "staticmap?center=41.5020952,-81.6789717&"
         + "zoom=14&size=350x450&sensor=false\"> ";

   //define an HTML page with code
   String myLocalHtmlPage =
          "<html> "
         +    "<body>"
         +     "<h1>Hello, world! </h1>"
         +     "<p> Greetings from <b>Cleveland State University</b>"
         +     "<p>" + aGoogleMapImage
         +    "</body> "
         + "</html>";

   webview.loadData( myLocalHtmlPage, "text/html", "UTF-8" );
}
```



WebViews

Example 1. MainActivity - (Demo3) Show Immediate Page

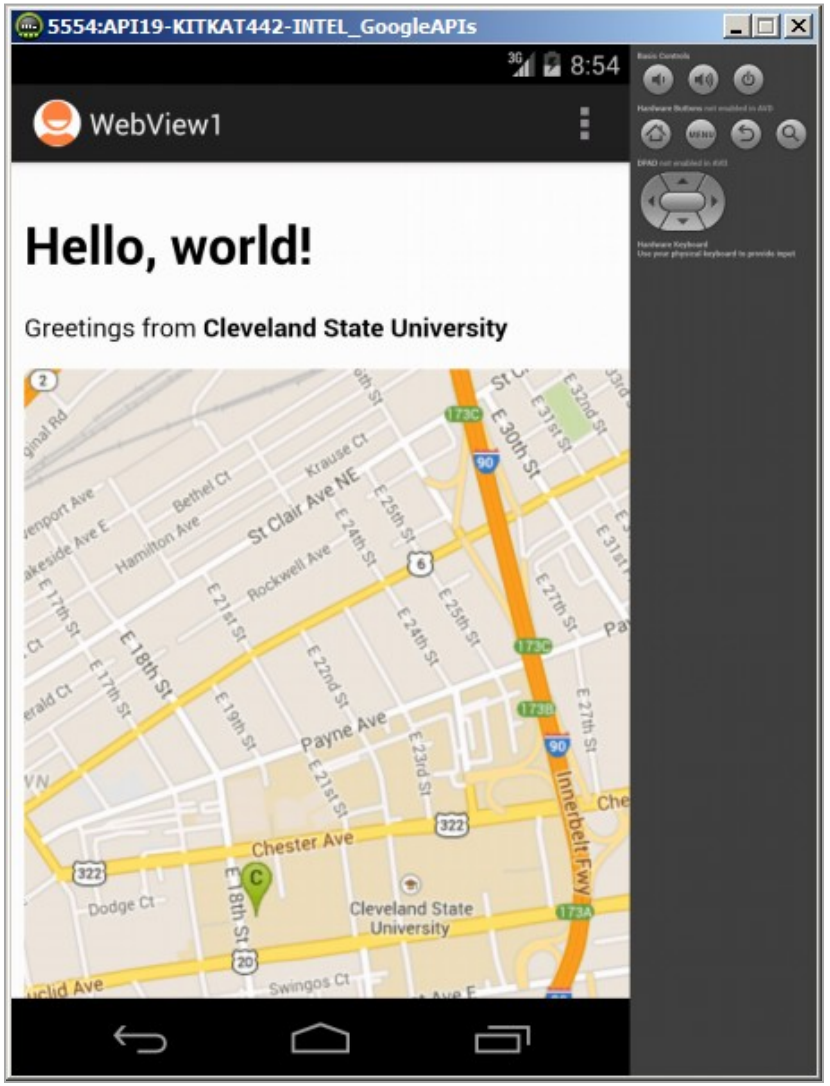
Warning! 

Make sure your app

1. Has the required **permission** clauses in the **Manifest** file (Internet and com.google.android.map library clauses are needed here)

2. Has been compiled using the Google.APIs library.

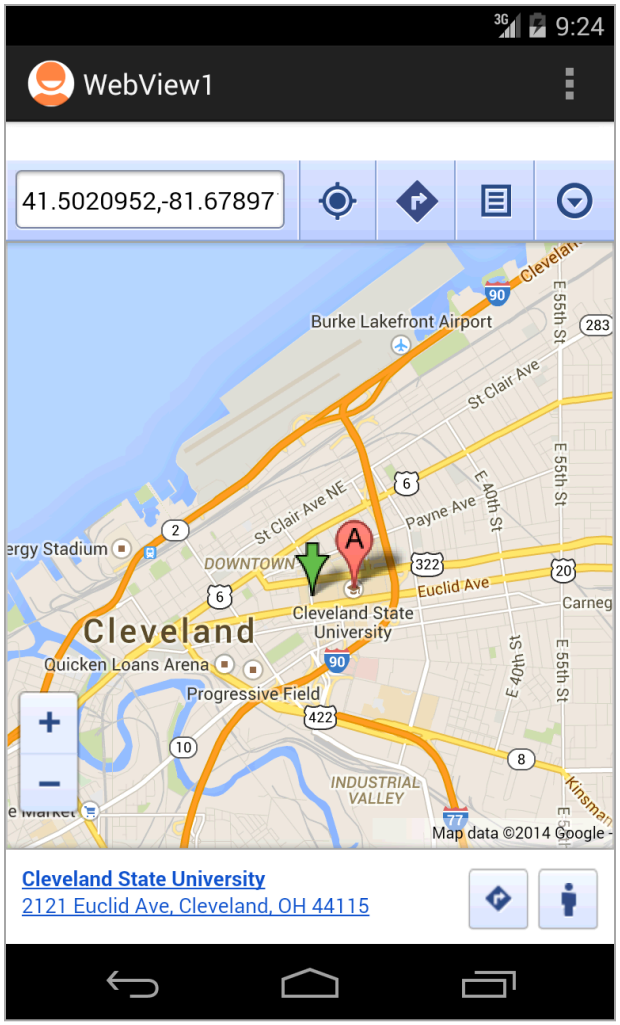
3. Your emulator is configured to target a GoogleAPIs environment (such as: *Google APIs (x86 System Image) (API Level 19*)



WebViews

Example 1. MainActivity - (Demo4) Show Google Map (Again)

Load a WebView with a URL defining a map according to **Google Map API V3**



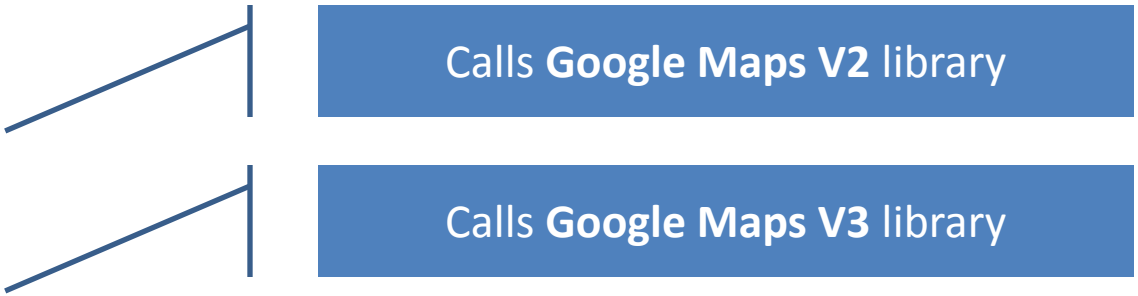

```
@SuppressLint("SetJavaScriptEnabled")
private void demo4TryRichGooggleMap() {
  webview = (WebView) findViewById(R.id.webView1);

  webview.getSettings().setJavaScriptEnabled(true);
  //show only WebViews
  webview.setWebViewClient(new WebViewClient());

  String mapUrl = "https://maps.google.com/maps"
          + "?q=41.5020952,-81.6789717&t=m&z=13";

  webview.loadUrl( mapUrl );

}
```

This example shows a 'rich' Google map with built-in controls for getting directions, zooming, street view, etc. For more information visit the link: https://developers.google.com/maps/documentation/javascript/basics#Mobile

WebViews

Example 1. MainActivity - (Demo4) Show Google Map (Again)

Note:
A subtle difference on the URLs from previous examples

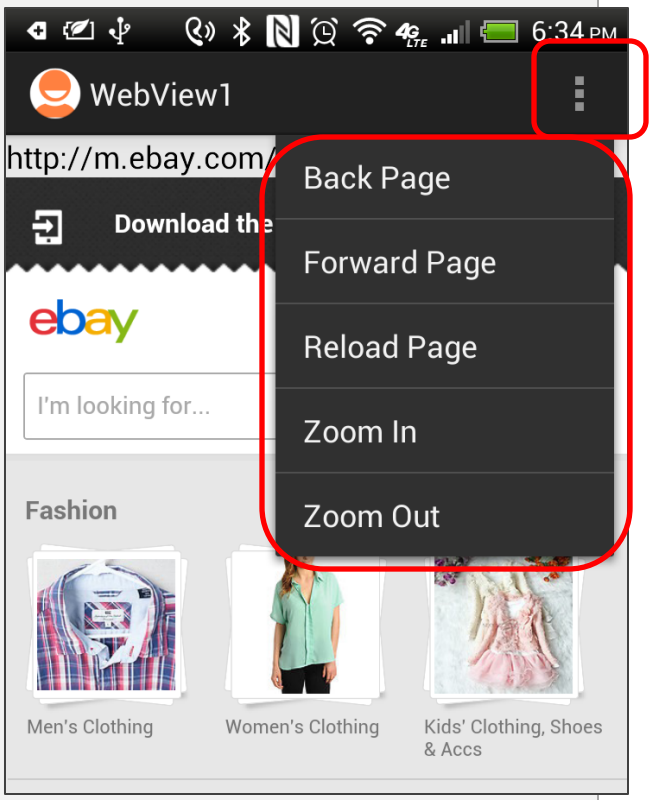


The V3 API is faster than V2 and is particularly designed with mobile devices on mind. In addition, the V2 API will not receive more *special* attention (see video https://www.youtube.com/v/zI8at1EmJjA)

WebViews

Example 1. MainActivity - Menu

```
    // browser navigation implemented through the option-menu
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_main_menu, menu);
        return true;
    }// onCreateOptionsMenu


    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        String option = item.getTitle().toString();
        if (option.equals("Forward Page"))
            webview.goForward();
        if (option.equals("Back Page"))
            webview.goBack();
        if (option.equals("Reload Page"))
            webview.reload();
        if (option.equals("Zoom In"))
            webview.zoomIn();
        if (option.equals("Zoom Out"))
            webview.zoomOut();
        return true;
    }// onOptionsItemSelected

}// class
```



Example 1. MyWebViewClient.java – Traffic Controller 1 of 2

```
class MyWebViewClient extends WebViewClient {
   // this object keeps a history-log of all visited links
   // and validates url links against a "home-base" server address
   Context context;
   TextView txtMsg;
   String hostServerSuffix; //this is the home-base

   public MyWebViewClient(TextView txtMsg, String hostServerSuffix) {
      this.txtMsg = txtMsg;
      this.hostServerSuffix = hostServerSuffix;
   }

   @Override
   public boolean shouldOverrideUrlLoading(WebView view, String url) {
      context = view.getContext();

      String host = Uri.parse(url).getHost();
      // if 'host' portion extracted above keeps us inside valid
      // base server (hostServerSuffix) then use a WebView to continue
      // navigation, otherwise override navigation by showing requested
      // page inside a full blown-up browser (security issues here...)

      String text = "URL:" + url.toString()
               + "\n\nHOST:" + host
               + "\n\nHOME should be:" + hostServerSuffix;
```

WebViews

Example 1. MyWebViewClient.java – Traffic Controller

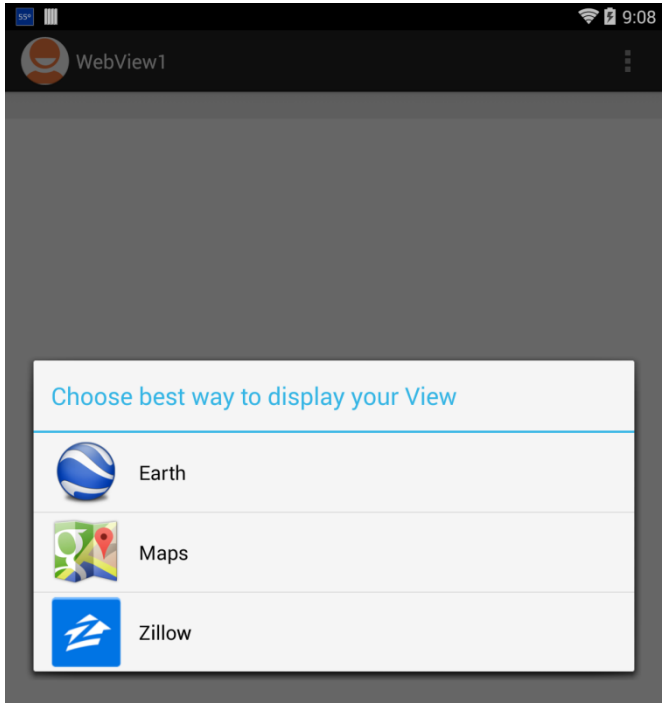
```
        Toast.makeText(context, text, Toast.LENGTH_LONG).show();

        if (url.contains(hostServerSuffix)) {
            // do not override navigation (using big web-browser) as long as
            // url points to some page in my defined home-server
            return false;

        } else {
            // The supplied url is not for a page on my 'valid' site,
            // in that case launch another Activity that handles URLs
            Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(url));
            context.startActivity(intent);
            return true;
        }
    }//shouldOverrideUrlLoading

    @Override
    public void onPageStarted(WebView view, String url, Bitmap favicon) {
        // keep user informed - update txtMsg with current URL value
        super.onPageStarted(view, url, favicon);
        txtMsg.setText(url.toString());
    }//onPageStarted

}//MyWebViewClient
```

Example 1. MyWebViewClient.java – Traffic Controller

What happens if no WebViewClient is supplied by the app ?

If a **WebViewClient** *is not provided, the* **WebView** will ask Activity Manager to choose the proper handler for the URL (the first time this happen an *App Chooser* screen will be displayed showing all candidate apps from which the user may select one)

The image on the right, shows a *chooser* for the mapping example Demo3. It lists apps registered to show a map , such as: Google-Earth, Google-Maps, And Zillow.



If a WebViewClient *is provided* it could inspect the given URL values.

- If the current address is 'acceptable' it returns FALSE, and the **WebView** assumes responsibility for displaying the contents of that page,
- otherwise the activity chooses the best way (usually a browser).

WebViews

Warning - Using JavaScript & Flash Animations

Your Android application **must** explicitly give permission to execute the **JavaScript** code of visited pages. By default WebViews have *Javascript* turned **off** . To activate JavaScripts do this:

```
webview.getSettings().setJavaScriptEnabled(true);
```

To remove the system's message warning you on the security risks brought by JS, add to each method enabling JS the following annotation

```
@SuppressLint("SetJavaScriptEnabled")
```

Warning:

Using JavaScript from visited pages creates a security dilemma; while JS is useful it is also dangerous (an attacker can include HTML that executes 'malevolent' code). As a rule: you should not allow JS unless you wrote all of the HTML and JavaScript that appears in our WebView (or you *trust* the visited site).

WebViews

Warning - Using JavaScript & Flash Animations

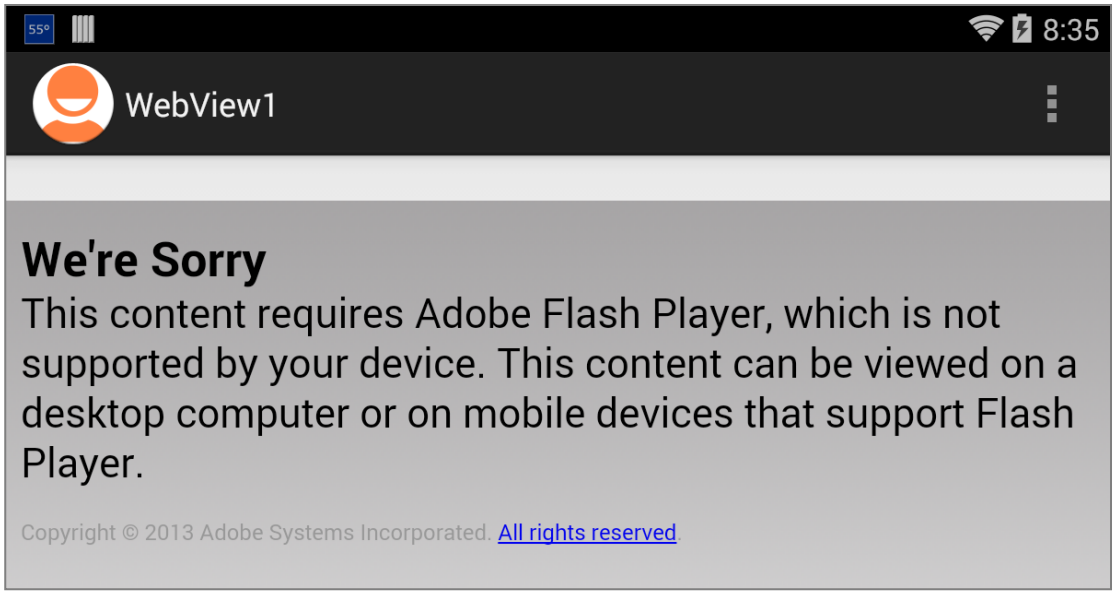
The use of plugins is deprecated and will not be supported in the future.
Do **not** use the following obsolete statement:

```
webview.getSettings().setPluginState(PluginState.ON);
```

In this example we tried to reach the **Adobe-Flash Download Site**, using the statement:

```
webview.loadUrl("http://get.adobe.com/flashplayer/");
```

Here the resulting image



Commonly Used WebView Methods

The following selected methods supplement the work of a WebView and make it look like a web-browser. These methods could be implemented through an Options-Menu or other GUI solution.

- **reload()** refresh the currently viewed web page
- **goBack()** go back one step in the browser history (use canGoBack() to determine if there is any history to trace back)
- **goForward()** go forward one step in the browser history (use canGoForward() to determine if there is any history to go forward to)
- **goBackOrForward()** go backwards or forwards in the browser history, where *negative/positive* numbers represent a count of steps to go
- **canGoBackOrForward()** to see if the browser can go backwards or forwards the stated number of steps.
- **clearCache()** clear the browser resource cache
- **clearHistory()** clear the browsing history
- **ZoomIn() , ZoomOut()** magnify control.

WebViews

Mixed Mode – Combining HTML & Android

Why?

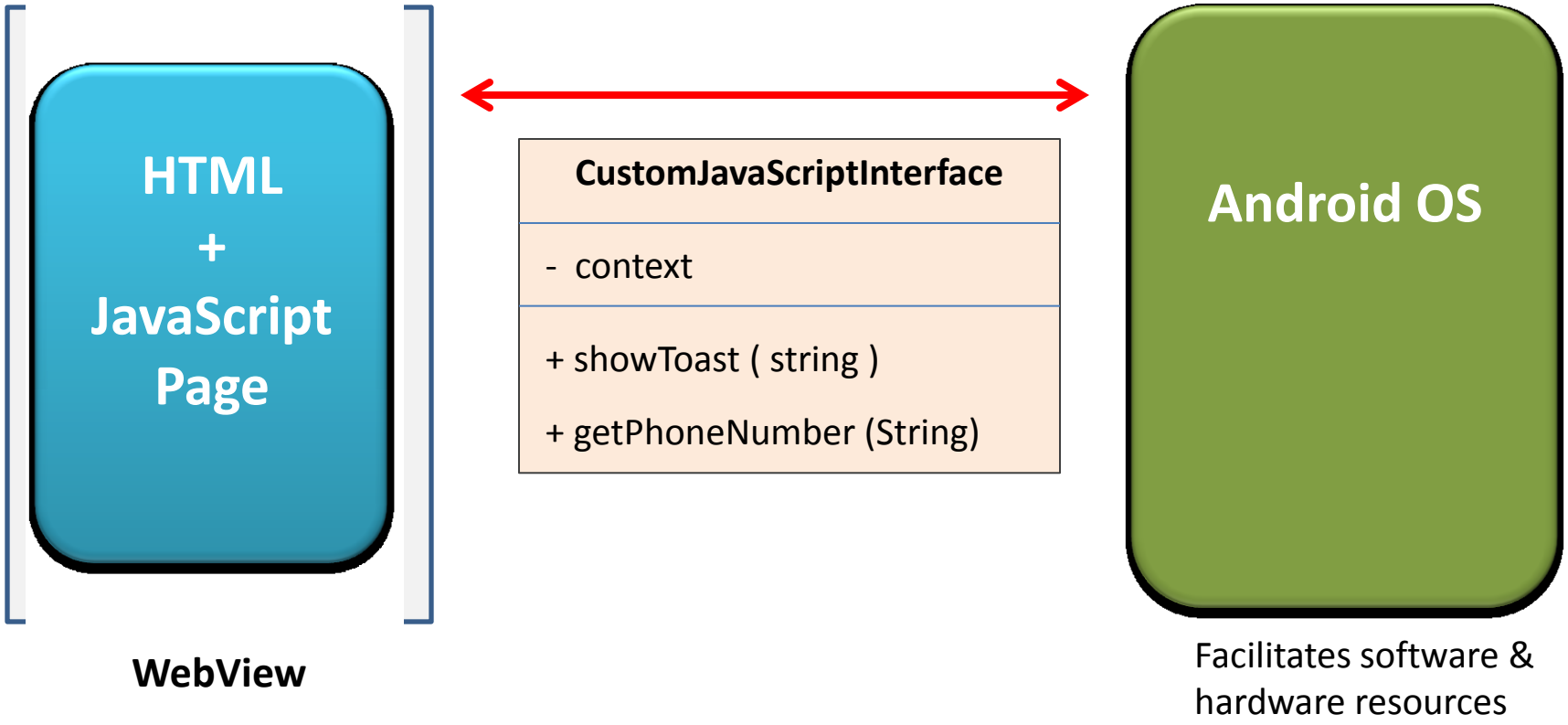
- HTML, CSS and JavaScript are very well understood, diffused and proven web-technologies.
- Many developers worldwide already know them.
- HTML-based applications are portable and -to a point- are hardware and software independent
- All mobile platforms (iOS, Android, Windows, Blackberry) support web page technology.
- Android offers through WebViews a simple way to expose HTML pages which are strongly connected to the OS.

How?

- GUI design and part of the business logic could be done with HTML+JS.
- JS supports the notion of a **JavaScriptInterface** acting as a bridge between the HTML and the Android sides of an app. Hardware and software resources found in the Android device could be presented to the HTML pages in a reasonably simple way.

WebViews

Mixed Mode – Combining HTML & Android



The **methods** in the interface are invoked by the JavaScript code placed behind the HTML side of the app. The interface methods run on the Android side of the app, using Android resources and data stored in the device.

WebViews

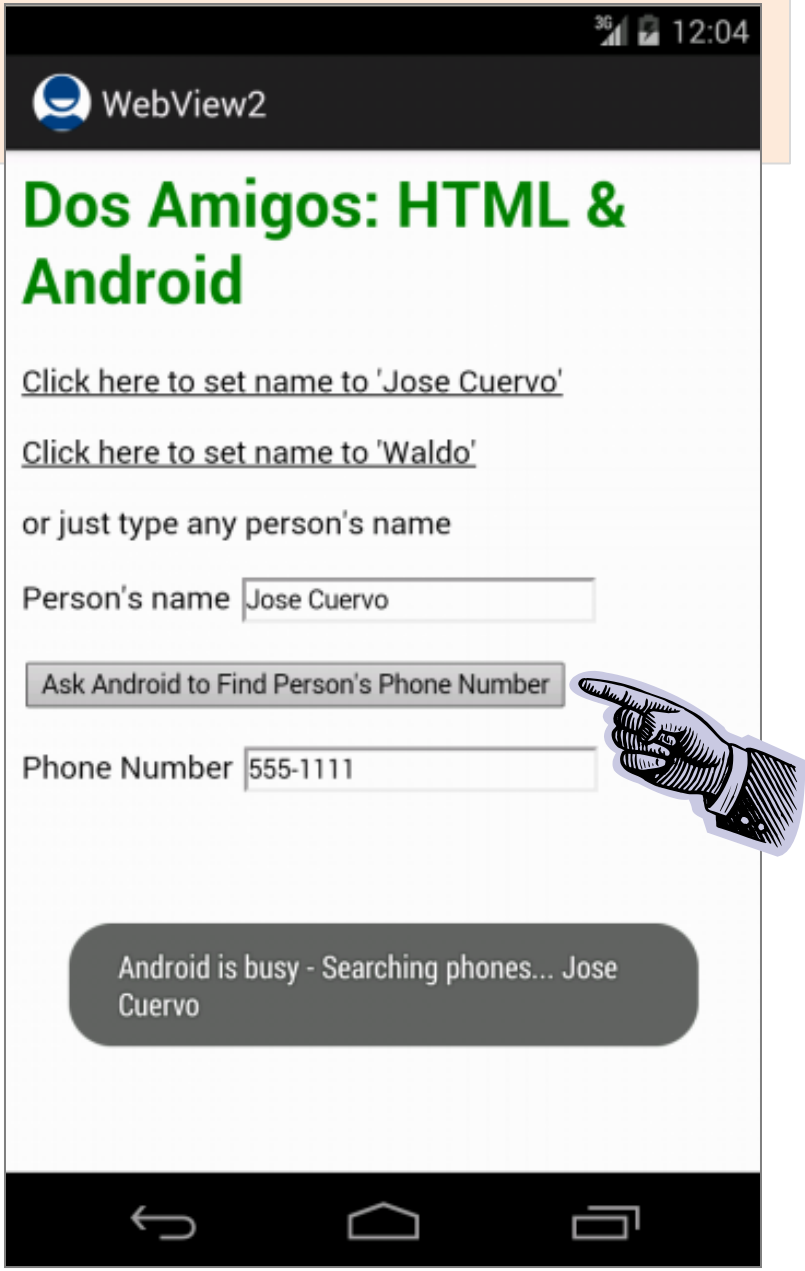
Example2. Mixed Mode – Combining HTML & Android

1. The app displays into a WebView an HTML page stored in the device.

2. The page accepts a person's name. After clicking the HTML button the page passes the person's name value to the server.

3. Android contributes to the app by searching its Contacts Database and returning the person's phone number.

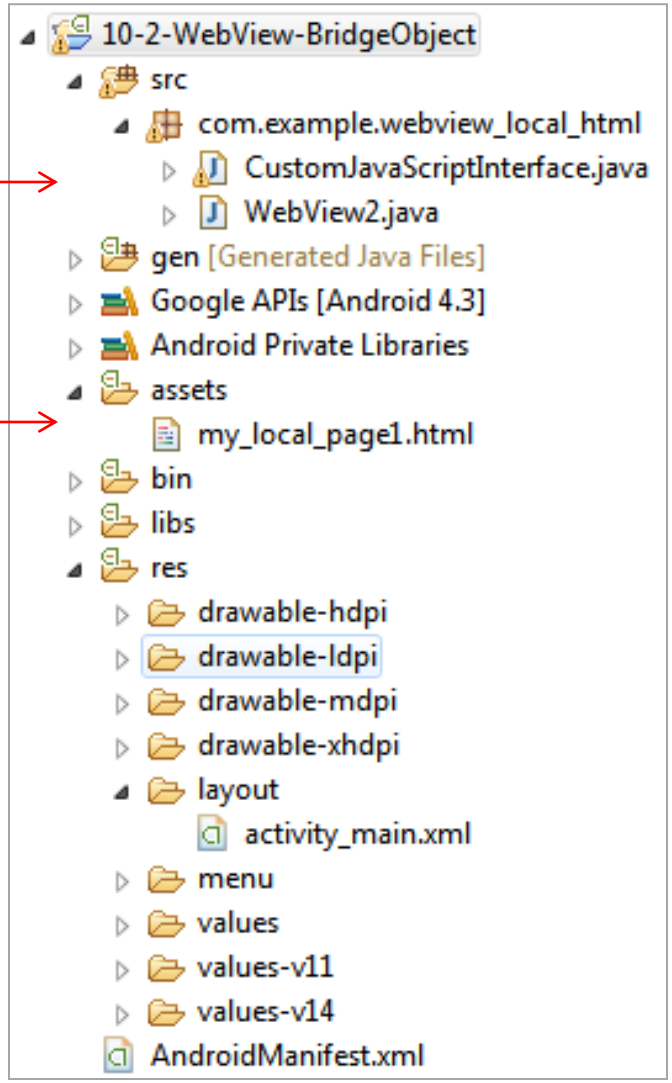
4. A 'pretty' Android Toast is used instead of a bland 'alert' JS-MessageBox



WebViews

Example2. Layout activity_main.xml

App's Structure




```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res
/android"
   xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <WebView
        android:id="@+id/webView1"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_alignParentLeft="true" />

</RelativeLayout>
```

WebViews

Example2. MainActivity - WebView2. java

```
public class WebView2 extends Activity {

   @SuppressLint("SetJavaScriptEnabled")
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      WebView webview = (WebView) findViewById(R.id.webView1);
      webview.getSettings().setJavaScriptEnabled(true);

      webview.addJavascriptInterface(new CustomJavaScriptInterface(this),
                                     "HtmlAndroidBridge");

      // if the HTML file is in the app's memory space use:
         webview.loadUrl("file:///android_asset/my_local_page1.html");

      // if the HTML files are stored in the SD card, use:
      // webview.loadUrl("file:///sdcard/my_local_webpage1.html");

      // CAUTION: Manifest must include
      // <uses-permission android:name="android.permission.INTERNET"/>
      // <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>

   }//onCreate

}//class
```

Example2. MainActivity - WebView2. java

Comments

CustomJavaScriptInterface is created as a Java class. An instance of this class is later passed to the JS environment through the statement:

```
webview.addJavascriptInterface(
                    new CustomJavaScriptInterface(this),
                  "HtmlAndroidBridge"  );
```

where `"HtmlAndroidBridge"` is the alias to be used by JS to designate elements in the object.

The JS code behind the HTML pages hosted by the WebView should refer to the methods in the bridge object using the notation

`HtmlAndroidBridge.methodName(args)`

HTML pages could be stored in either the application's memory space or the larger SD card.

Example2. CustomJavaScriptInterface.java

```
public class CustomJavaScriptInterface {
    private Context context;

    CustomJavaScriptInterface(Context context) {
        // remember the application's context
        this.context = context;
    }

    @JavascriptInterface
    public void showToast(String toastMsg) {
        // instead of dull JavaScript alert() use flashy Toasts
        Toast.makeText(context, toastMsg, Toast.LENGTH_SHORT).show();
    }

    @JavascriptInterface
    public String getPhoneNumber(String person) {
        // accept a person's name and fake that you are
        // searching the Android's Contacts Database
        person = person.toLowerCase();
        if (person.equals("jose cuervo"))
            return "555-1111";
        else if (person.equals("waldo"))
            return "555-2222";
        else
            return "555-3333";
    }
}
```

1 → `@JavascriptInterface` / `public void showToast(String toastMsg)`

2 → `@JavascriptInterface` / `public String getPhoneNumber(String person)`

See the appendix for a realistic Android function to search the Contact List looking for a given person.

Example2. CustomJavaScriptInterface.java

Comments

1. The public method **showToast(string)** is called by JS code to briefly display an arbitrary message on top of the HTML page held by the WebView. Observe that **Toast** is an Android object not an HTML resource.

2. The method **getPhoneNumber(string)** is used to pretend that an Android search into the Contact's database is taking place. For simplicity we have chosen to limit the number of contacts to only three people. However, in the appendix you will find a more appropriate solution. Databases and Content Providers are studied later in the course.

WebViews

Example2. HTML – my_local_page1.html 1 of 2

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
 <head>
 <title>Android GetPhone Demo</title>
 <script language="javascript">

   function setPerson(nameValue) { // move nameValue to input box – reset phone
      document.getElementById("guiName").value = nameValue ;
      document.getElementById("guiPhone").value = "" ;
   }

   function askAndroidToFindPhone() {
      // call Android, ask it to find the person's phone number
      var person = document.getElementById("guiName").value;
1 →   HtmlAndroidBridge.showToast('Android is busy - Searching phones... '
                                  + person);
      document.getElementById("guiPhone").value =
2 →                                  HtmlAndroidBridge.getPhoneNumber(person);
   }
</script>
 <style type="text/css">
 .style1 {
    color: #008000;
    font-family: Arial, Helvetica, sans-serif;
 }
 </style>
</head>
```

Example2. HTML – my_local_page1.html 2 of 2

```
<body>
  <h1 class="style1">Dos Amigos: HTML & Android</h1>

  <p><a onClick="setPerson('Jose Cuervo')">
     <u> Click here to set name to 'Jose Cuervo'</u></a>

  <p><a onClick="setPerson('Waldo')">
     <u> Click here to set name to 'Waldo'</u></a>

  <p>or just type any person's name

  <p> Person's name  <input type="text" id="guiName" />

❸ <p> <input type="button" onclick= "askAndroidToFindPhone()"
            value="Ask Android to Find Person's Phone Number">

  <p> Phone Number   <input type="text" id="guiPhone" />

 </body>

</html>
```

Example2. HTML – my_local_page1.html

Comments

1. The HTML page assembles a visual interface exposing among others an input TextBox for the user to enter a person's name. Clicking a button activates a search for the person's phone number.

2. Like in any common HTML page the tag `<script language="javascript">` is used to add code-behind support. JS scripts can occasionally implement all of the business-logic associated to the app; however the typical case is that most solutions are obtained by collaboration.

3. The shared object **HtmlAndroidBridge** allows JS code-behind to call Android methods. This is needed whenever JS wants to reach hardware and software resources that are available in the Android powered device (observe that JS has no direct access to them).

4. Notice how HTML pages can be enhanced using CSS specifications.

Example3. Embedding Existing Web Technology in Android

In the next Android app example we will add mapping services obtained from **Google Maps JavaScript API V3.** This API brings new features to mapping operations . it extends the capabilities shown in Example1-Demo3 and Demo4 in which Google maps are requested by a browser or WebView as part of an URL.

Why?

Advantages offered by Android

1. Access to native hardware & software resources on the device (such as fine location services, contact list, music, pictures, etc)
2. 'Easy' placement of the finished app in the Android Market.
3. Good tools for rapid development using the Android SDK and Eclipse.

Advantages offered by Google Maps Service

1. Rapid versioning of the server-held software. This eliminates the requirement to download and install frequent Android-app updates.
2. More frequent feature additions and bug fixes from Google.
3. Cross-platform compatibility: Using the Maps API allows you to create a single map that runs on multiple platforms.

Example3. Embedding Existing Web Technology in Android

What do we want?

- Map a given location using **Google Maps JavaScript API V3**. (https://developers.google.com/maps/documentation/javascript/)
- The map is expected to show its most current and updated features.
- The mapping effort should be the same but the final product must adapt itself to the current hardware powered by Android.

How to do it?

- First, use your desktop and a browser to prototype the map. Write a simple HTML document that shows an image obtained from calling Google Maps API with the appropriate geo-coding information.
- Next, create an Android app that displays a **WebView**.
- Load the **HTML+JS** page design above, into the WebView (Done!)

Example3. Embedding Existing Web Technology in Android

What is Exactly Google Maps JavaScript API V3 ?

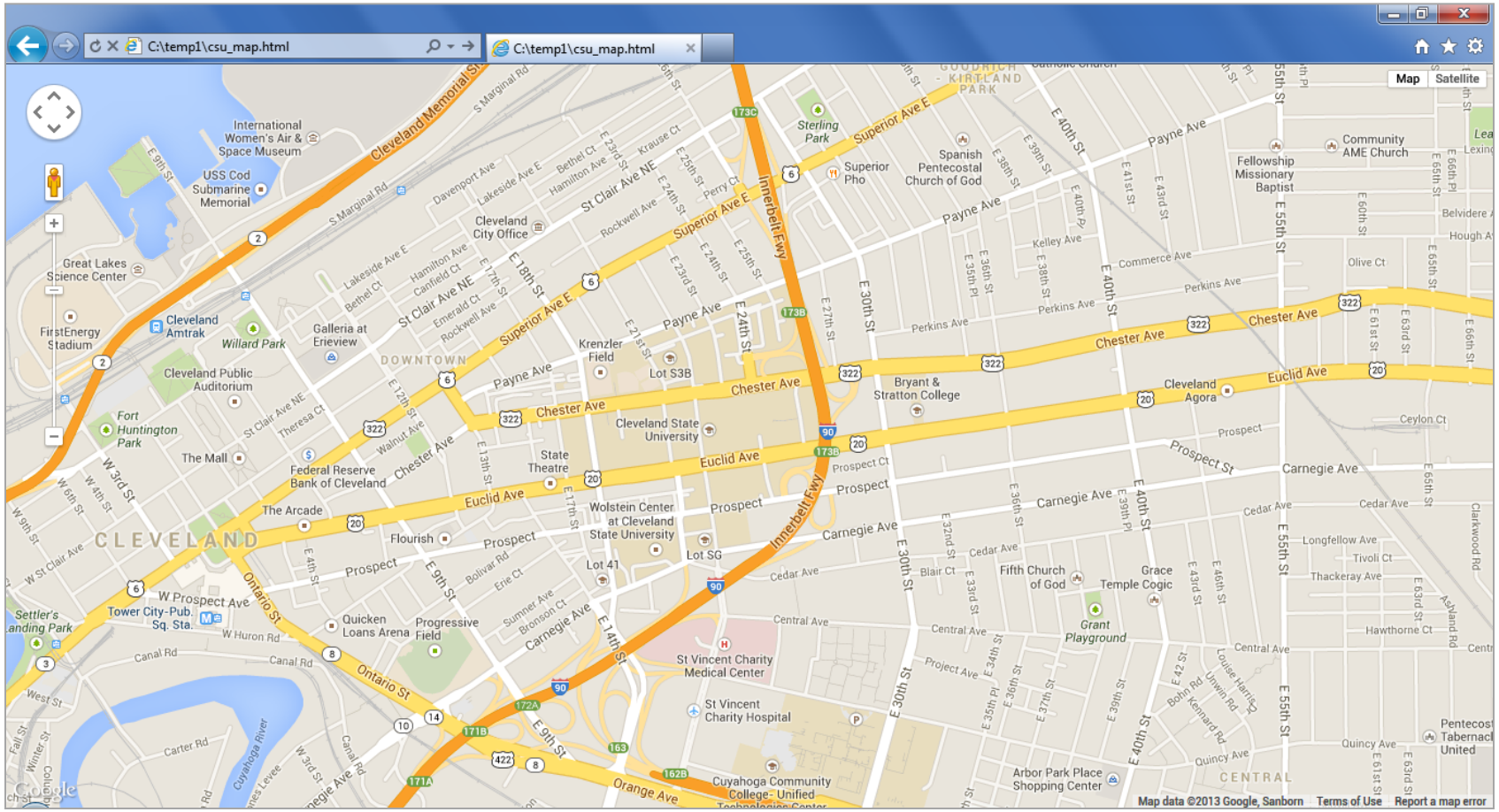
- The **Google Maps Javascript API** is a free service that lets you embed Google Maps in your own web pages.
- It is especially designed to be faster and more applicable to mobile devices (as well as traditional desktop browser applications)
- The API provides a number of utilities for manipulating maps (just like on the http://maps.google.com web page) and adding content to the map through a variety of services, allowing you to create rich maps applications on your website/app.

Link: https://developers.google.com/maps/documentation/javascript/

Example3. Embedding Existing Web Technology in Android

Map on a Windows Browser

The image below shows a Google Map created with the API V3. It's centered around "Cleveland State University, Ohio" (IExplorer on a Windows machine)



Link: https://developers.google.com/maps/documentation/javascript/basics?csw=1

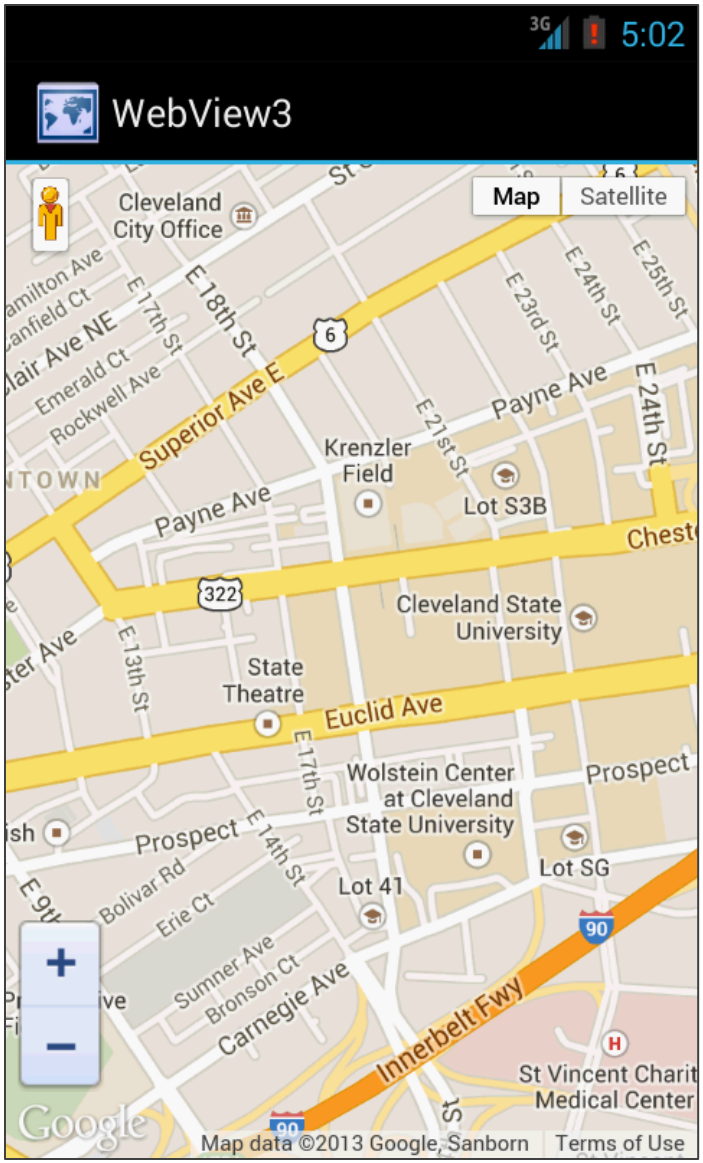
Example3. Embedding Existing Web Technology in Android

Map on an Android Device

The image shows the same Google map depicted in the previous page ('Cleveland State University, Ohio').

This time the image comes from an Android emulator running an app that uses a WebView to show the HTML page.

As in the previous page, the map includes scrolling, zooming, road & satellite modes, as well as street view capabilities.



WebViews

Example3. HTML + Javascript + API V3

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />
1 → <style type="text/css">
      html { height: 100% }
      body { height: 100%; margin: 0px; padding: 0px }
      #map_canvas { height: 100% }
  </style>
2 → <script type="text/javascript"
          src="http://maps.google.com/maps/api/js?sensor=false">
  </script>
  <script type="text/javascript">
3 →  function initialize() {
     var latlng = new google.maps.LatLng(41.5020952, -81.6789717);
     var myOptions = {
        zoom: 15,
        center: latlng,
        mapTypeId: google.maps.MapTypeId.ROADMAP
     };
     var map = new google.maps.Map(document.getElementById("map_canvas"), myOptions);
   }
  </script>
  </head>
4 → <body onload="initialize()">
       <div id="map_canvas" style="width:100%; height:100%"></div>
  </body>
</html>
```

google_map.html

This HTML page uses a JavaScript function to set a Google Map around a specific set of coordinates (CSU Student Center)

Example3. Embedding Existing Web Technology in Android

Comments

google_map.html is the HTML page responsible for drawing the Google Map.

1. The **<style>** tag sets the entire screen (100% height, width) as the canvas on which the map is to be drawn.

2. The first **<script>** fragment calls an API method to draw a map. Observe that no actual location is supplied (the map is not displayed yet).

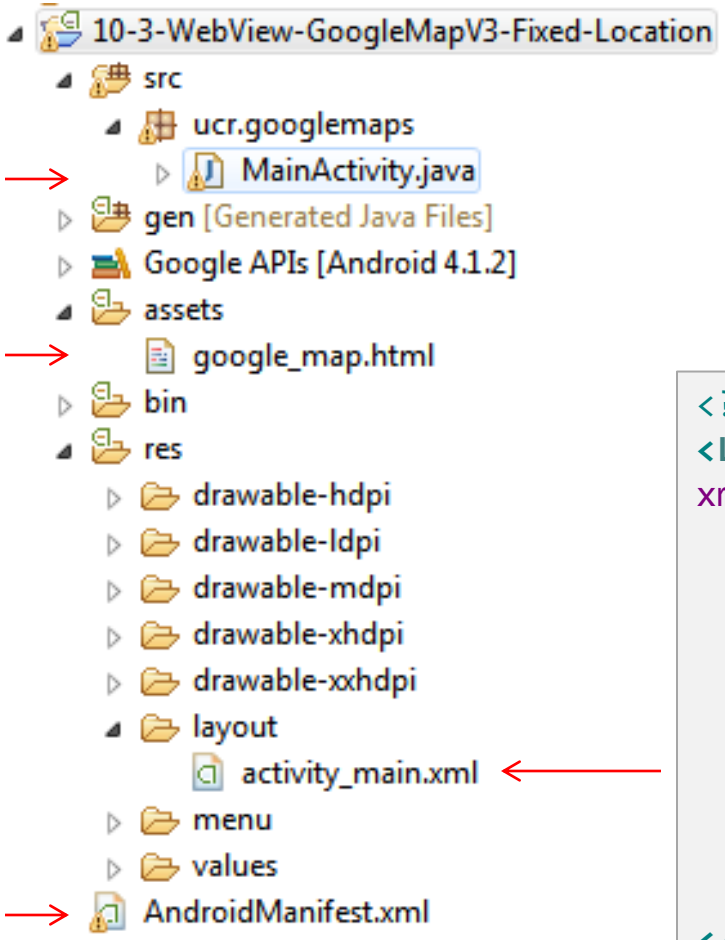
3. The **Initialize** JS function sets mapping arguments. Map is to be centered around the supplied latitude and longitude. ROADMAP mode (other options are SATELLITE and TRAFFIC), as well as zooming factor (15 out of 21) are requested.

4. When the HTML page is loaded the **Initialize** function is called and the map is created.

WebViews

Example3. Embedding Existing Web Technology in Android

Using Google JavaScript Maps API V3



Putting the pieces together:

1. Place a **WebView** in the activity_main.xml file
2. Copy the previous HTML page in the **assets** folder
3. Add **Internet** and **maps permission** to the app's manifest
4. Done

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

  <WebView
      android:id="@+id/webview"
      android:layout_width="match_parent"
      android:layout_height="match_parent"/>

</LinearLayout>
```

WebViews

Example3. Embedding Existing Web Technology in Android

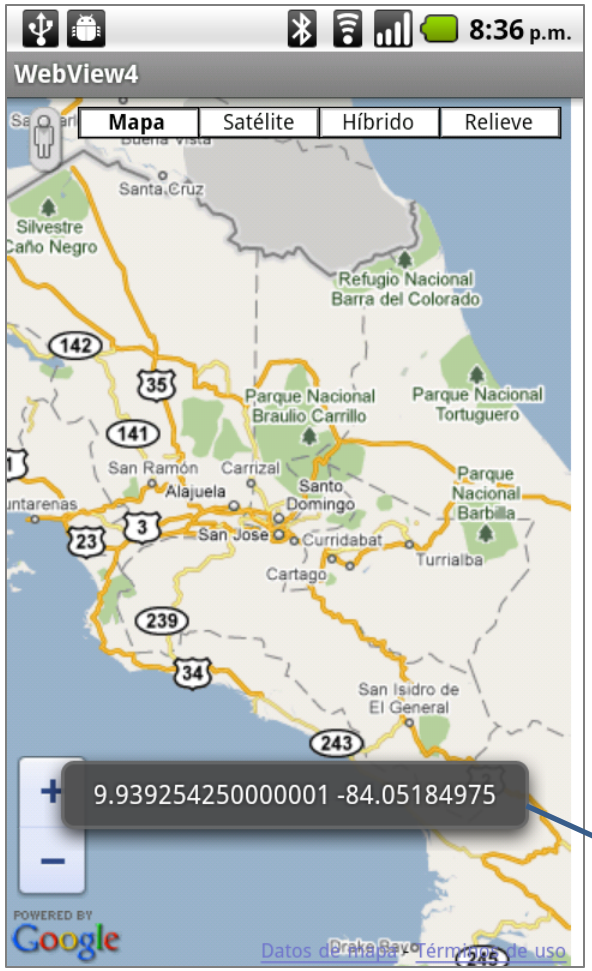
```
public class MainActivity extends Activity {
    WebView webview;

    @SuppressLint("SetJavaScriptEnabled")
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // load into WebView local HTML page defining mapping operation
        webview = (WebView) findViewById(R.id.webview);
        webview.getSettings().setJavaScriptEnabled(true);

        webview.loadUrl("file:///android_asset/google_map.html");

    }//onCreate

}//class
```

The Android side of the app loads an HTML page and does no more work. The app's intelligence is written in the HTML + JS page(s).

Delay until **Service** Lesson

Example4. Cross-Platform Collaboration

Using Google JavaScript Maps API V3 (using **real** locations)



This experience combines the two previous examples:

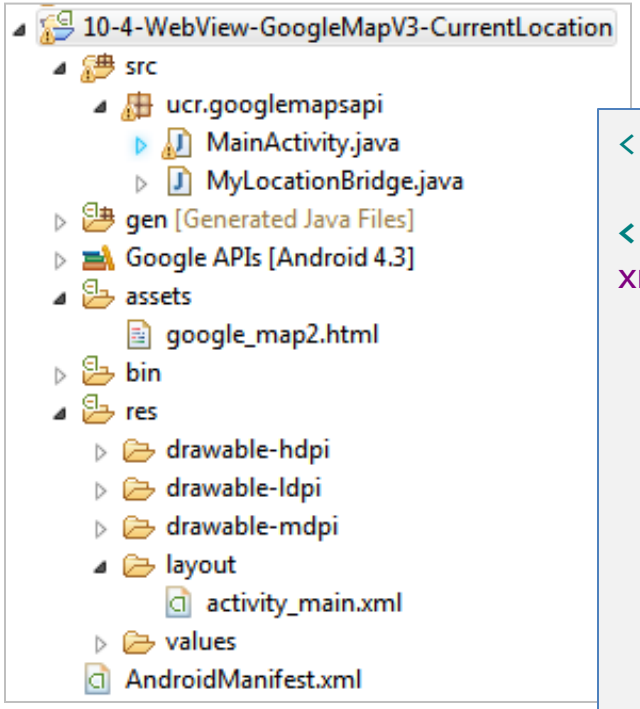
- Android's GPS hardware finds the actual location of the device.
- A JavaScript Bridge-Object passes the *real location* data to an HTML webpage.
- The page contains a JavaScript function to draw a map centered on the given coordinates.

Latitude and longitude detected by the device's GPS chip. Image taken from an Android phone.

WebViews

Example4. Cross-Platform Collaboration

App Structure



LAYOUT - activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

  <WebView
      android:id="@+id/webview"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"/>

</LinearLayout>
```

This example is based on:
http://code.google.com/apis/maps/articles/android_v3.html
http://code.google.com/apis/maps/documentation/javascript/overlays.html#MarkerAnimations
http://gmaps-samples.googlecode.com/svn/trunk/articles-android-webmap

Example4. MainActivity.java


```
public class MainActivity extends Activity implements LocationListener {
   // a LocationListener could use GPS, Cell-ID, and WiFi to establish
   // a user's location. Deciding which to use is based on choices
   // including factors such as accuracy, speed, and battery-efficiency.

   private WebView webview;
     LocationManager locationManager;
     MyLocationBridge locationBridge = new MyLocationBridge();

   @Override
   protected void onDestroy() {
      super.onDestroy();
      // cut location service requests
      locationManager.removeUpdates(this);
   }

   private void getLocation() {
      locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
      Criteria criteria = new Criteria();
      // criteria.setAccuracy(Criteria.ACCURACY_FINE); // use GPS(you should be outside)
      criteria.setAccuracy(Criteria.ACCURACY_COARSE);  // cell towers, wifi
      String provider = locationManager.getBestProvider(criteria, true);

      // In order to make sure the device is getting the location, request
      // updates [wakeup after changes of: 5 sec. or 10 meter]
      locationManager.requestLocationUpdates(provider, 5, 10, this);
      locationBridge.setNewLocation(locationManager.getLastKnownLocation(provider));
   }
```

Example4. MainActivity.java


```
@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
   getLocation();
   setupWebView();
   this.setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
}//onCreate

// Set up the webview object and load the page's URL
@SuppressLint("SetJavaScriptEnabled")
private void setupWebView() {
   final String centerMapURL = "javascript:centerAt("
                             + locationBridge.getLatitude() + ","
                             + locationBridge.getLongitude() + ")";
   // set up the webview to show location results
   webview = (WebView) findViewById(R.id.webview);
   webview.getSettings().setJavaScriptEnabled(true);
   webview.addJavascriptInterface(locationBridge, "locationBridge");
   webview.loadUrl("file:///android_asset/google_map2.html");
   // Wait for the page to load then send the location information
   webview.setWebViewClient(new WebViewClient() {
      @Override
      public void onPageFinished(WebView view, String url) {
         webview.loadUrl(centerMapURL);
      }
    });
  } //setUpWebView
```

Example4. MainActivity.java


```
    @Override
    public void onLocationChanged(Location location) {
        String lat = String.valueOf(location.getLatitude());
        String lon = String.valueOf(location.getLongitude());
        Toast.makeText(getApplicationContext(), lat + "\n" + lon, 1).show();
        locationBridge.setNewLocation(location);
    }

    @Override
    public void onProviderDisabled(String provider) {
        // needed by Interface. Not used
    }

    @Override
    public void onProviderEnabled(String provider) {
        // needed by Interface. Not used
    }

    @Override
    public void onStatusChanged(String provider, int status, Bundle extras) {
        // needed by Interface. Not used
    }


}//class
```

Example4. MainActivity.java


```
// An object of type "MyLocationBridge" will be used to pass data back and
// forth between the Android app and the JS code behind the HTML page.

public class MyLocationBridge {
     private Location mostRecentLocation;

      @JavascriptInterface
     public void setNewLocation(Location newCoordinates){
          mostRecentLocation = newCoordinates;
     }
      @JavascriptInterface
     public double getLatitude() {
          if (mostRecentLocation == null) return (0);
          else return mostRecentLocation.getLatitude();
     }
      @JavascriptInterface
     public double getLongitude() {
          if (mostRecentLocation == null) return (0);
          else return mostRecentLocation.getLongitude();
     }

}// MyLocationFinder
```

Example4. google_map2.html

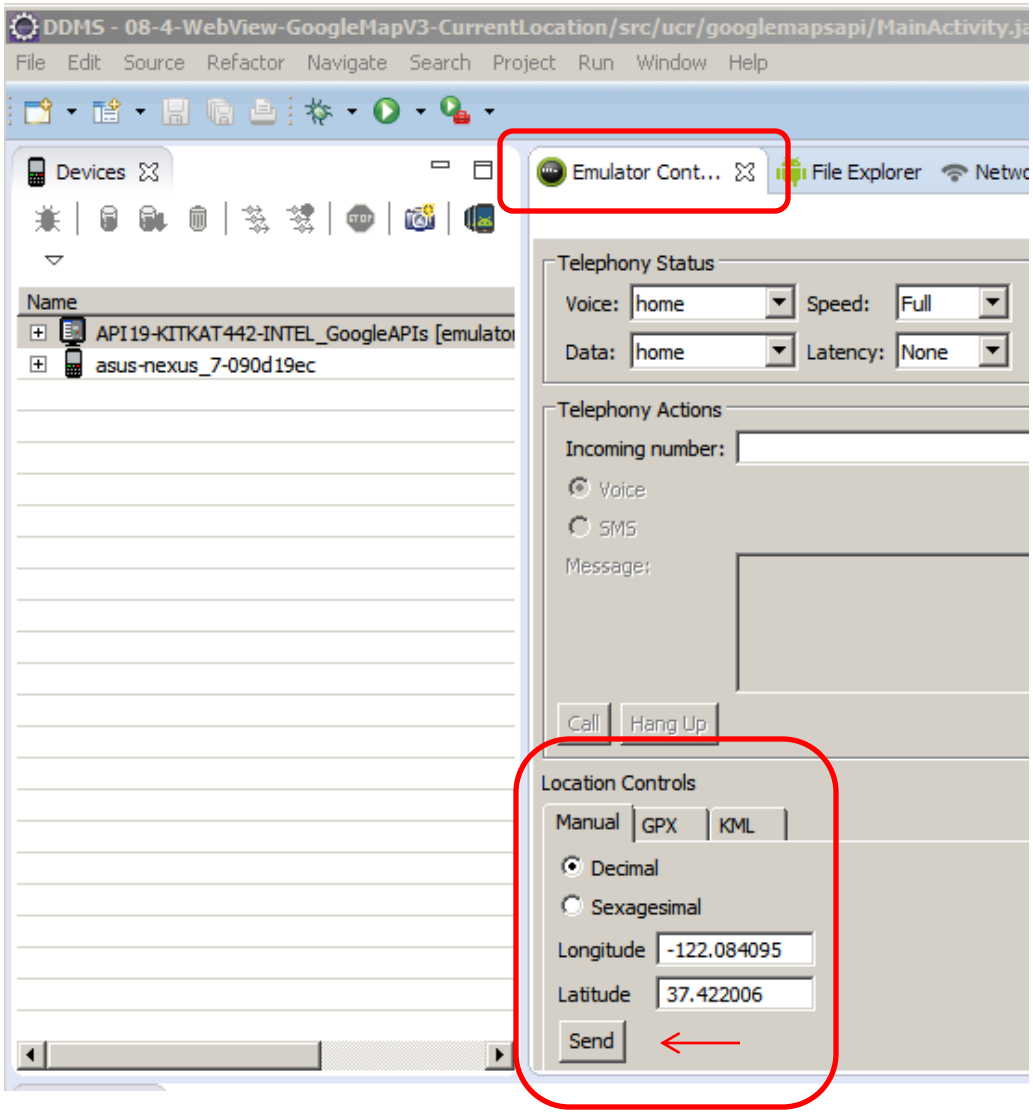

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />

<meta http-equiv="content-type" content="text/html; charset=UTF-8"/>

<title>Google Maps JavaScript API v3 Example: Marker Simple</title>

<style type="text/css">
   html { height: 100% }
   body { height: 100%; margin: 0px; padding: 0px }
   #map_canvas { height: 100% }
</style>

<script type="text/javascript"
        src="http://maps.google.com/maps/api/js?sensor=false">
</script>
```

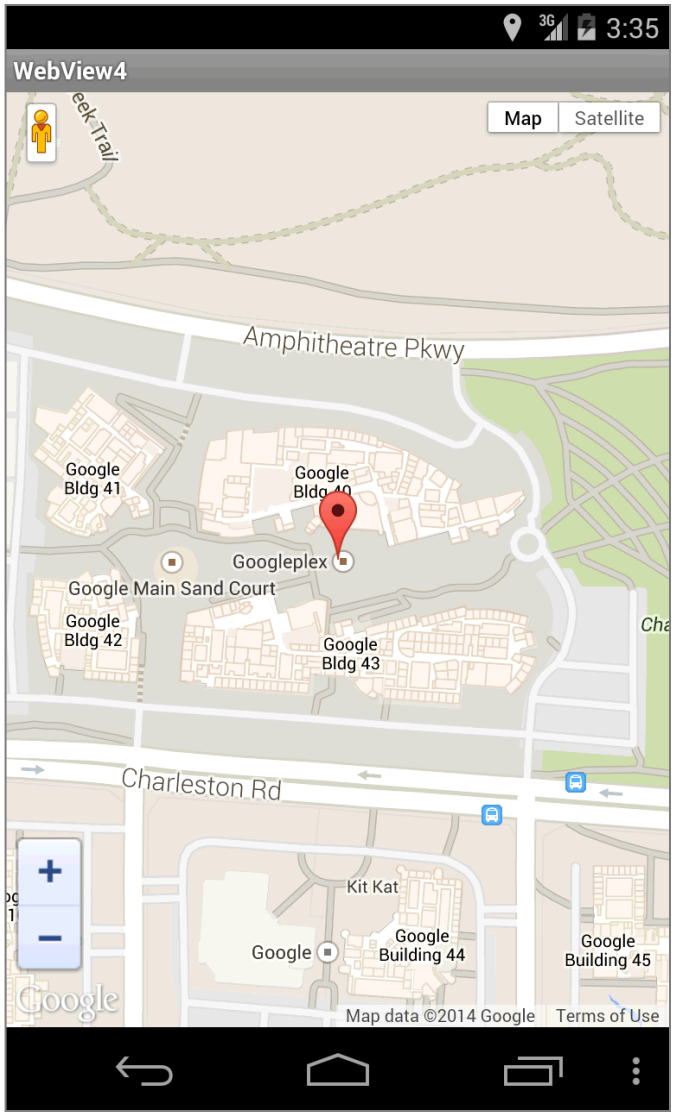
Example4. google_map2.html


```
<script type="text/javascript">
function initialize() {
   // get latitude and longitude from the JavaScripInterface
   var myLatlng = new google.maps.LatLng( locationBridge.getLatitude(),
                                          locationBridge.getLongitude());
   var myOptions = {
         zoom: 17,
         center: myLatlng,
         mapTypeId: google.maps.MapTypeId.ROADMAP
   }
   var map = new google.maps.Map(document.getElementById("map_canvas"),
                                 myOptions);

    var marker = new google.maps.Marker({
        position: myLatlng,
        map: map
    });
  }
</script>

</head>
<body onload="initialize()">
  <div id="map_canvas"></div>
</body>
</html>
```

WebViews

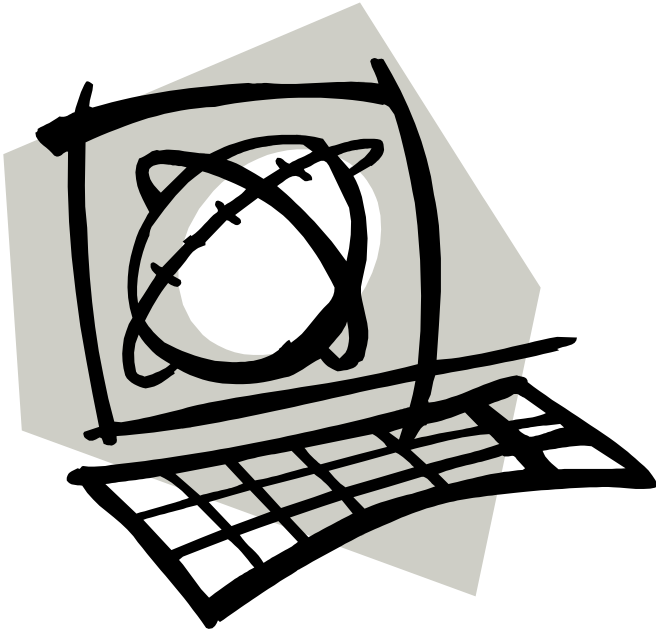
Example4. Testing App with the Emulator Control





Android's WebViews

Questions ?



APPENDIX A. A List of suggested material on Android and Web Technologies

Oracle Mobile Application Framework
http://www.oracle.com/technetwork/developer-tools/maf/overview/index.html
How to build mobile apps for Android and iOS using Java-based Oracle MAF Platform

Intel App Framework
http://app-framework-software.intel.com/
JavaScript Library for Mobile HTML5 Multi-Platform Development

Performance Tips for Geo API (55 min video – explains why G-maps V3 API was created)
https://www.youtube.com/v/zI8at1EmJjA

Google Maps Developers Documentation
https://developers.google.com/maps/documentation/

Google Maps JavaScript API V3 Tutorial
http://code.google.com/apis/maps/documentation/javascript/tutorial.html

Building Web Apps in WebView
http://developer.android.com/guide/webapps/webview.html
How to embed web pages into your Android application using WebView and bind JavaScript to Android APIs.

Debugging Web Apps
http://developer.android.com/guide/webapps/debugging.html
How to debug web apps using JavaScript Console APIs.

Best Practices for Web Apps
http://developer.android.com/guide/webapps/best-practices.html
Practices you should follow in order to provide an effective web application on Android-powered devices.

APPENDIX B. A realistic (although very inefficient!!!) function to search for a phone number using the user's Contact-Book

Add to Manifest

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

```
public String findPhoneNumber(String searchName) {
  //look for first contact entry partially matching searchName

  String name = "n.a.";
  String number = "n.a.";
  //defina an iterator to traverse the contact book
  Cursor phones = getContentResolver().query(
         ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
         null, null,  null, null);

  while (phones.moveToNext()) {
      String NAME = ContactsContract.CommonDataKinds.Phone.DISPLAY_NAME;
      name = phones.getString(phones.getColumnIndex(NAME));
      String NUMBER = ContactsContract.CommonDataKinds.Phone.NUMBER;
      number = phones.getString(phones.getColumnIndex(NUMBER));

      if (name.toLowerCase().contains(searchName.toLowerCase())) {
        return number; // a good match, phone number found
    }
  }
  // Not found
  return "n.a.";
}// findPhoneNumber
```