Android Development

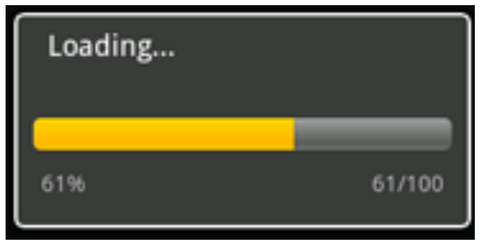
Lecture 5
Android Graphical User Interface 3

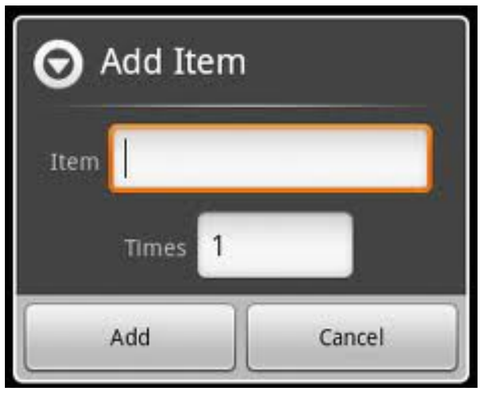
Lecture Summary

- Toast Notifications
- Dialogs
- WebView
- Web Client / Web Chrome Client
- Load Local Web Content
- WebView Javascript Interface
- Supporting Multiple Screens

Dialogs

- A dialog is a small window that appears in front of the current Activity.
- The underlying Activity loses focus and the dialog accepts all user interaction. Dialogs are used for notifications that should interrupt the user and to perform short tasks that directly relate to the application in progress (such as a progress bar or a login prompt).
- The Dialog class is the base class for creating dialogs. You can also use one of the following subclasses:
  - AlertDialog
  - ProgressDialog
  - DatePickerDialog
  - TimePickerDialog
- If you would like to customize your own dialog, you can extend the base Dialog object or any of the subclasses listed above and define a new layout.

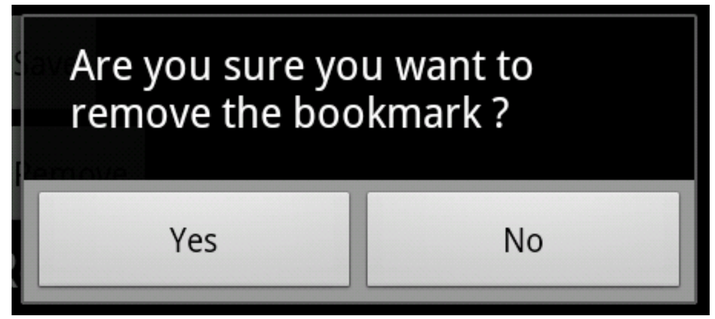




Alert Dialog

- A dialog that can manage zero, one, two, or three buttons, and/or a list of selectable items that can include checkboxes or radio buttons. The AlertDialog is capable of constructing most dialog user interfaces and is the suggested dialog type.

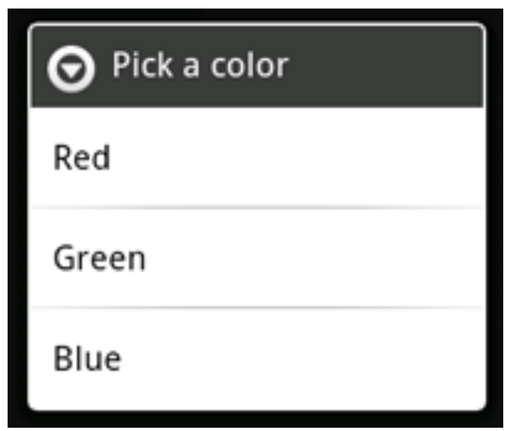
```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setMessage("Are you sure you want to exit?")
       .setCancelable(false)
       .setPositiveButton("Yes", new DialogInterface.OnClickListener() {
           public void onClick(DialogInterface dialog, int id) {
                MyActivity.this.finish();
           }
       })
       .setNegativeButton("No", new DialogInterface.OnClickListener() {
           public void onClick(DialogInterface dialog, int id) {
                dialog.cancel();
           }
       });
AlertDialog alert = builder.create();

alert.show();
```



Alert Dialog

- It is also possible to create an AlertDialog with a list of selectable items using the method setItems().

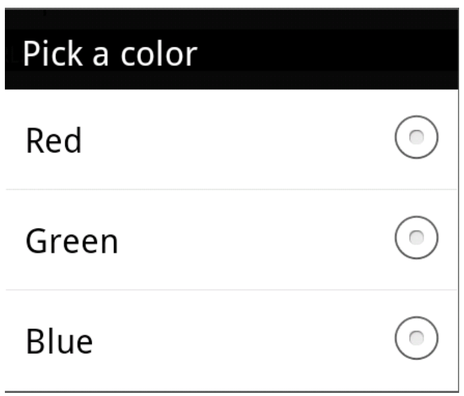
```
final CharSequence[] items = {"Red", "Green", "Blue"};

AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setTitle("Pick a color");
builder.setItems(items, new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int item) {
        ...
    }
});
AlertDialog alert = builder.create();
```



Alert Dialog

- Using the setMultiChoiceItems() and setSingleChoiceItems() methods, it is possible to create a list of multiple-choice items (checkboxes) or single-choice items (radio buttons) inside the dialog.

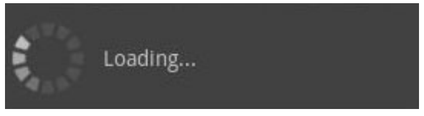
```
final CharSequence[] items = {"Red", "Green", "Blue"};

AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setTitle("Pick a color");
builder.setSingleChoiceItems(items, -1, new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int item) {

    }
});
AlertDialog alert = builder.create();
```



Progress Dialog

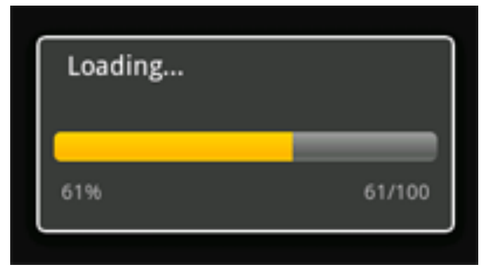
- A ProgressDialog is an extension of the AlertDialog class that can display a progress animation in the form of a spinning wheel, for a task with progress that's undefined, or a progress bar, for a task that has a defined progression.

```
ProgressDialog dialog = ProgressDialog.show(MyActivity.this, "", "Loading. Please wait...", true);
```



- If you want to create a progress bar that shows the loading progress with granularity you can :

```
ProgressDialog progressDialog;
progressDialog = new ProgressDialog(mContext);
progressDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
progressDialog.setMessage("Loading...");
progressDialog.setCancelable(false);
...
progressDialog.show();
```



- You can increment the amount of progress displayed in the bar by calling either setProgress(int) with a value for the total percentage completed so far or incrementProgressBy(int) with an incremental value to add to the total percentage completed so far. You can also specify the maximum value in specific cases where the percentage view is not useful.

Custom Dialog

- It is also possible to customize the design of a dialog. You can create your own layout for the dialog window with layout and widget elements.
- When the Dialog has been instantiated you can set your custom layout as the dialog's content view with setContentView(int), passing it the layout resource ID.
- Using the method findViewById(int) on the dialog object it is possible to retrieve and modify its content.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:id="@+id/layout_root"
              android:orientation="horizontal"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:padding="10dp"
              >
    <ImageView android:id="@+id/image"
               android:layout_width="wrap_content"
               android:layout_height="fill_parent"
               android:layout_marginRight="10dp"
               />
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="fill_parent"
              android:textColor="#FFF"
              />
</LinearLayout>
```

```
Context mContext = getApplicationContext();
Dialog dialog = new Dialog(mContext);

dialog.setContentView(R.layout.custom_dialog);
dialog.setTitle("Custom Dialog");

TextView text = (TextView)
dialog.findViewById(R.id.text);
text.setText("Hello, this is a custom dialog!");
ImageView image = (ImageView)
dialog.findViewById(R.id.image);
image.setImageResource(R.drawable.android);
```

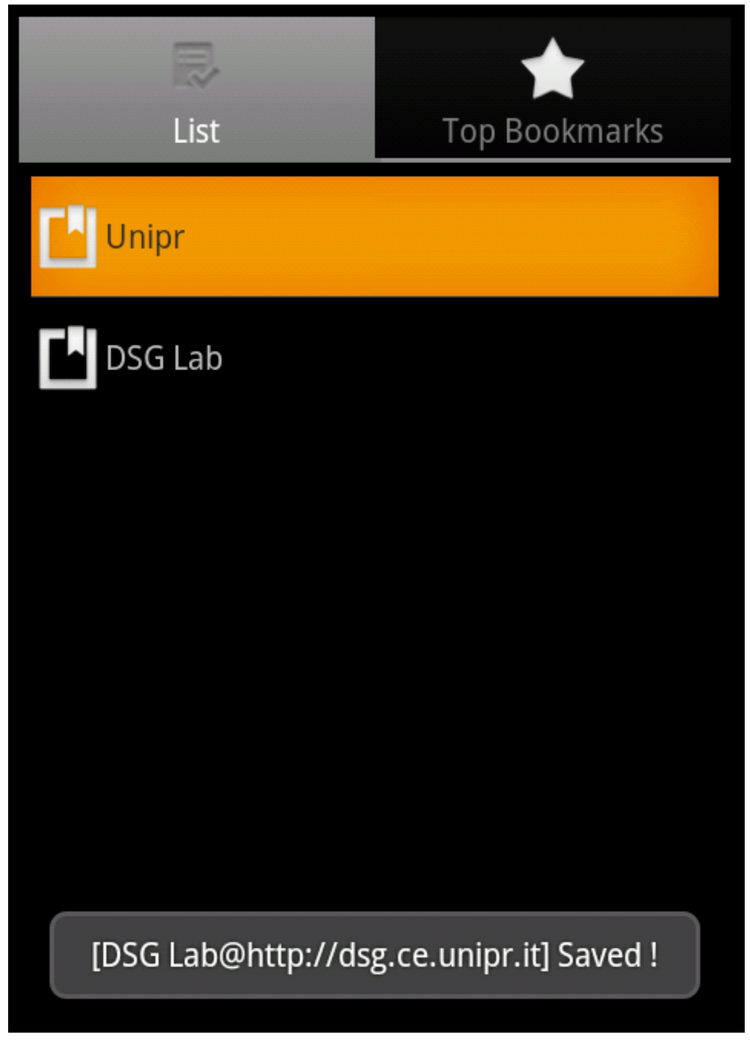
Toast Notifications

- A toast notification is a message that pops up on the surface of the active view filling the amount of space required for the message.

- The current activity remains visible and interactive.

- The notification automatically fades in and out, and does not accept interaction events.

- A toast notification can be created and displayed from an Activity or Service. If you create a toast notification from a Service, it appears in front of the Activity currently in focus.



Toast Notifications

- First, instantiate a Toast object with one of the makeText() methods. This method takes three parameters: the application Context, the text message, and the duration for the toast.
- It returns a properly initialized Toast object and using the show() method you can display it.

```
Context context = getApplicationContext();
CharSequence text = "Hello toast!";
int duration = Toast.LENGTH_SHORT;

Toast toast = Toast.makeText(context, text, duration);
toast.show();
```

- The same result can be obtained in a short and easy way using a single line command

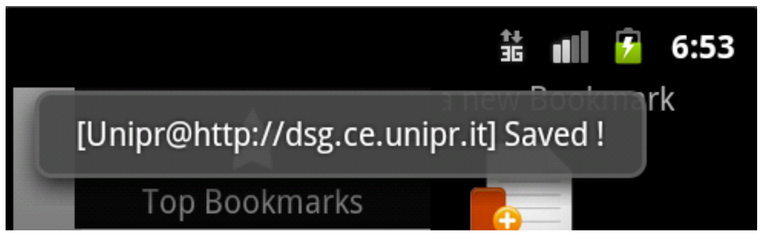
```
Toast.makeText(context, "Hello toast!", Toast.LENGTH_SHORT).show();
```

Toast Notifications - Position

- A standard toast notification appears near the bottom of the screen, centered horizontally. You can change this position with the setGravity(int, int, int) method. This accepts three parameters: a Gravity constant, an x-position offset, and a y-position offset.

```
toast.setGravity(Gravity.TOP|Gravity.LEFT, 0, 0);
```

- For example, with the previous code you decide that the toast will appear in the top left corner.



Custom Toast Notification

- There is also the possibility to create a customized layout for your toast notification.
- As for previous UI element you can define custom layout by defining a View layout, in XML or in your application code, and pass the root View object to the setView(View) method.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
android"
              android:id="@+id/toast_layout_root"
              android:orientation="horizontal"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:padding="10dp"
              android:background="#DAAA"
              >
    <ImageView android:id="@+id/image"
               android:layout_width="wrap_content"
               android:layout_height="fill_parent"
               android:layout_marginRight="10dp"
               />
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="fill_parent"
              android:textColor="#FFF"
              />
</LinearLayout>
```

- Notice that the ID of the LinearLayout element is "toast_layout". You must use this ID to inflate the layout from the XML.

Custom Toast Notification

- To build a custom toast using a custom xml layout you can retrieve the LayoutInflater with getLayoutInflater() (or getSystemService()), and then inflate the layout from XML using inflate(int, ViewGroup).
- The first parameter is the layout resource ID and the second is the root View.
- You can use this inflated layout to find more View objects in the layout, so now capture and define the content for the ImageView and TextView elements.
- Finally, create a new Toast with Toast(Context) and set some properties of the toast. Then call setView(View) and pass it the inflated layout. You can now display the toast with your custom layout by calling show().

```
LayoutInflater inflater = getLayoutInflater();
View layout = inflater.inflate(R.layout.toast_layout,(ViewGroup) findViewById(R.id.toast_layout_root));

ImageView image = (ImageView) layout.findViewById(R.id.image);
image.setImageResource(R.drawable.android);
TextView text = (TextView) layout.findViewById(R.id.text);
text.setText("Hello! This is a custom toast!");

Toast toast = new Toast(getApplicationContext());
toast.setGravity(Gravity.CENTER_VERTICAL, 0, 0);
toast.setDuration(Toast.LENGTH_LONG);
toast.setView(layout);
toast.show();
```

Web View

- If you want to deliver a web application (or just a web page) as a part of a client application, you can do it using WebView.
- The WebView class is an extension of Android's View class that allows you to display web pages as a part of your activity layout. It does not include any features of a fully developed web browser, such as navigation controls or an address bar. All that WebView does, by default, is show a web page.
- Common scenarios in which you can use a WebView are:
  - to provide information in your application that you might need to update, such as an end-user agreement or a user guide or generally content that requires an internet connection to retrieve data.
  - to use code that has been already written in HTML/Javascript (Maps, Registration Form, etc ...).
  - to present a local web page that's tailored for Android devices to show dedicated content.
  - to show a web link retrieved by your application (for example from Facebook or Twitter) without starting the OS Browser but keeping the user inside your app.

Web View

- To add a WebView to your Application, simply include the <WebView> element in your activity layout. The you can load a web page in the WebView, use loadUrl() and if it is necessary you can also enable the support for Javascript code.

```
<?xml version="1.0" encoding="utf-8"?>
<WebView  xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/webview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
/>
```

```
WebView myWebView = (WebView) findViewById(R.id.webview);
webSettings.setJavaScriptEnabled(true);
myWebView.loadUrl("http://www.example.com");
```

- If the WebView load a remote content you need to add to your Android Manifest the Internet Permission.

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

Web View - Handling Page Navigation

- When the user clicks a link from a web page in your WebView, the default behavior is for Android to launch an application that handles URLs. Usually, the default web browser opens and loads the destination URL.
- You can override this behavior for your WebView, so links open within your WebView. You can then allow the user to navigate backward and forward through their web page history that's maintained by your WebView.
- To have a complete control of the WebView you can implement your own WebViewClient that overrides required methods such as the shouldOverrideUrlLoading().

```
private class BookmarkWebViewClient extends WebViewClient {
      @Override
      public boolean shouldOverrideUrlLoading(WebView view, String url) {
          view.loadUrl(url);
          return true;
      }

      public void onReceivedError(WebView view, int errorCode, String description, String failingUrl) {
          ...
        }
  }                    WebView myWebView = (WebView) findViewById(R.id.webview);
                       myWebView.setWebViewClient(new BookmarkWebViewClient());
```

Web View - Page History

- When your WebView overrides URL loading, it automatically accumulates a history of visited web pages. You can navigate backward and forward through the history with goBack() and goForward().
- For example you can add dedicated button to go back or forward in you activity or you can use the standard BackButton to handle page navigation. To use the back button to navigate backward you should override the method onKeyDown in the the activity containing the WebView.

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    // Check if the key event was the Back button and if there's history
    if ((keyCode == KeyEvent.KEYCODE_BACK) && myWebView.canGoBack() {
        myWebView.goBack();
        return true;
    }
    // If it wasn't the Back key or there's no web page history, bubble up to the default
    // system behavior (probably exit the activity)
    return super.onKeyDown(keyCode, event);
}
```

WebViewClient & WebChromeClient

- As showed in the previous slide, the WebViewClient allows to monitor and control the behavior of the WebView associated to events that have an impact to the rendering of the content such as errors, form submissions or URL loading.
- On the other hand The WebChromeClient subclass allows to manage the behavior of the WebView when things happen that might impact a browser UI happens, for instance, progress updates and JavaScript alerts or logs.

```
private class BookmarkWebChromeClient extends WebChromeClient {
    @Override
    public void onProgressChanged(WebView view, int progress) {
      Log.d(MainActivity.TAG,"BookmarkWebChromeClient ---> onProgressChanged: " + progress);
    }

  @Override
  public void onConsoleMessage(String message, int lineNumber,String sourceID) {
     super.onConsoleMessage(message, lineNumber, sourceID);
     Log.d(MainActivity.TAG,"BookmarkWebChromeClient ---> onConsoleMessage: " + message);
  }
}
```

WebChromeClient Compatibility

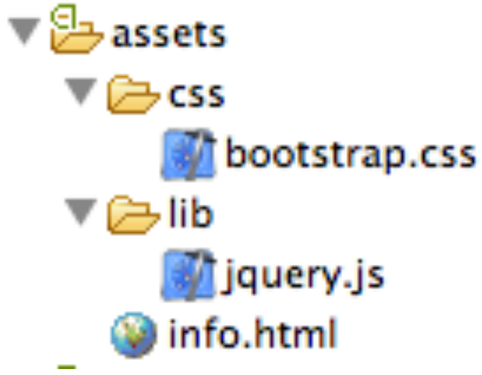
```
private class BookmarkWebChromeClient extends WebChromeClient {
    @Override
    public void onProgressChanged(WebView view, int progress) {
      Log.d(MainActivity.TAG,"BookmarkWebChromeClient ---> onProgressChanged: " + progress);
      progressBar.setProgress(progress);
      if(progress == 100) {
         progressBar.setVisibility(View.GONE);
           }
    }
    @Override
    public boolean onConsoleMessage(ConsoleMessage consoleMessage) {
       Log.d(MainActivity.TAG,"BookmarkWebChromeClient ---> onConsoleMessage: " + consoleMessage.message());
       return super.onConsoleMessage(consoleMessage);
    }

    /*OLD METHOD THAT NOW IS DEPRECATED
    @Override
    public void onConsoleMessage(String message, int lineNumber,String sourceID) {
       super.onConsoleMessage(message, lineNumber, sourceID);
       Log.d(MainActivity.TAG,"BookmarkWebChromeClient ---> onConsoleMessage: " + message);
    }
    */
}
```

WebView Load Local Content

- A WebView object can be used also to load a HTML string using loadData method. In this case you need to specify the content type and the encoding (null to use the default one).

```
String summary = "<html><body>You scored <b>192</b> points.</body></html>";
webview.loadData(summary, "text/html", null);
```

- loadUrl method can be also used to load a local HTML content from a resource folder. Android offers one more directory where you can keep files which also will be included in package. This directory called assets. The difference between /res and /assets is that Android doesn't generate IDs for assets content. You can access to this folder using AssetManager assetManager = getAssets(); or using a direct url "file:///android_asset/".




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

Binding Javascript and Android Code

- It is possible to create an interface between your JavaScript code and client-side Android code when you are developing a web application using a WebView on Android.
- For example, your JavaScript code can call a method in your Android code to display a Dialog, or can call a method to retrieve specific data or information showing them in a Web page.
- To bind a new interface between your JavaScript and Android code, call addJavascriptInterface(), passing it a class instance to bind to your JavaScript and an interface name that your JavaScript can call to access the class.

```
public class MyJavaScriptInterface {
    Context mContext;
    /** Instantiate the interface and set the context */
    MyJavaScriptInterface(Context c) {
        mContext = c;
    }
    /** Show a toast from the web page */
    public void showToast(String toast) {
        Toast.makeText(mContext, toast, Toast.LENGTH_SHORT).show();
    }
}
```

Binding Javascript and Android Code

- You can bind this class to the JavaScript that runs in your WebView with addJavascriptInterface() and name the interface Android.

```
WebView webView = (WebView) findViewById(R.id.webview);
webView.addJavascriptInterface(new JavaScriptInterface(this), "Android");
```

- When the interface has been created and the Web content is correctly loaded, your web application has access to the JavaScriptInterface class.

```
<input type="button" value="Say hello" onClick="showAndroidToast('Hello Android!')" />

<script type="text/javascript">
    function showAndroidToast(toast) {
        Android.showToast(toast);
    }
</script>
```

- The WebView automatically initialize the Android interface making it available to your web page. So, at the click of the button, the showAndroidToast() function uses the Android interface to call the JavaScriptInterface.showToast() method.

Binding Security & Compatibility

- An application can inject Java objects into a WebView via the addJavascriptInterface function.
- This allows JavaScript code to call the public methods of the injected Java object.
- Exposing Java objects to JavaScript could have some negative security implications, such as allowing JavaScript to invoke native phone functionality (sending SMS to premium numbers, accessing account information, etc.) or allowing JavaScript to subvert existing browser security controls such as the same origin policy.
- Applications targeted to API level 17 (Android 4.2), and above in the future, protect against reflection-based attack by requiring programmers to annotate exposed functions (@JavascriptInterface)

http://www.cis.syr.edu/~wedu/Research/paper/webview_acsac2011.pdf

Binding Security & Compatibility

```
public class MyJavaScriptInterface {
    Context mContext;

    /** Instantiate the interface and set the context */
    public MyJavaScriptInterface(Context c) {
        mContext = c;
    }

    /** Show a toast from the web page */
    @JavascriptInterface
    public void showToast(String toast) {
     Log.d(MainActivity.TAG,"JavaScriptInterface ---> showToast !");
        Toast.makeText(mContext, toast, Toast.LENGTH_SHORT).show();
    }

    @JavascriptInterface
    public String getData() {
     Log.d(MainActivity.TAG,"JavaScriptInterface ---> showToast !");
        return "<p>Stored Bookmarks: " + BookmarkManager.getInstance().getBookmarkList().size() +"</p>";
    }
}
```

Binding Javascript and Android Code

- **Note**: "*The object that is bound to your JavaScript runs in another thread and not in the thread in which it was constructed*".

- **Caution**:
"***Using addJavascriptInterface() allows JavaScript to control your Android application.***
This can be a very useful feature or a dangerous security issue. When the HTML in the WebView is untrustworthy (for example, part or all of the HTML is provided by an unknown person or process), then an attacker can include HTML that executes your client-side code and possibly any code of the attacker's choosing. As such, you should not use addJavascriptInterface() unless you wrote all of the HTML and JavaScript that appears in your WebView.
You should also not allow the user to navigate to other web pages that are not your own, within your WebView (instead, allow the user's default browser application to open foreign links—by default, the user's web browser opens all URL links, so be careful only if you handle page navigation as described in the following section)".

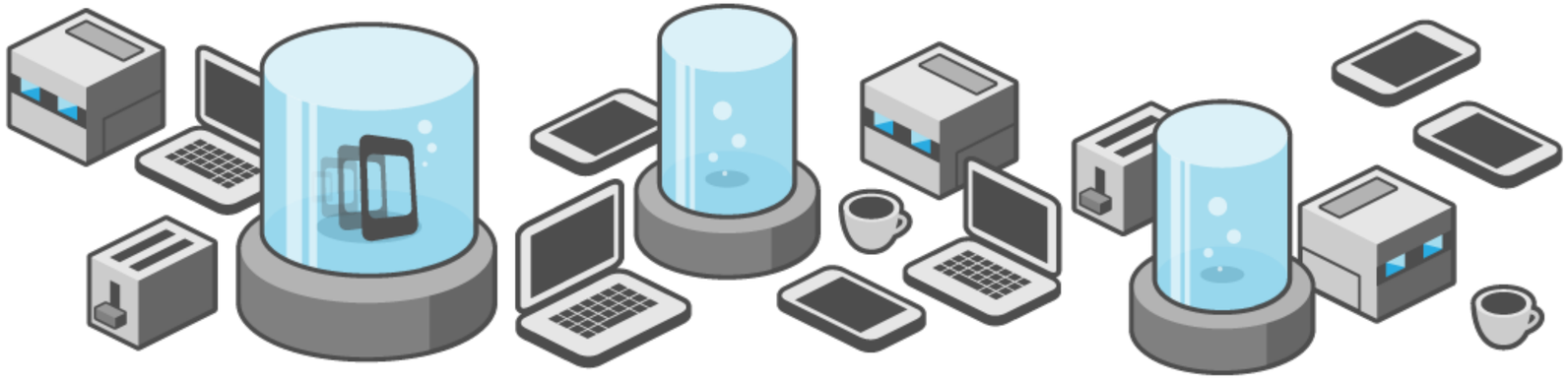
Binding Javascript and Android Code

- On Android 2.3.x Emulator a call from Javascript of a Java method through javascript binding generate an error. The same problem it is not present on previous and next Emulator versions.
- If your application uses a bind between Javascript and Android Code you can in any case test your application on a real device or on an emulator != 2.3.x

```
04-12 01:09:12.548: I/Web Console(466): Showing Applications Statistics .. at file:///android_asset/info.html:28
04-12 01:09:12.548: W/dalvikvm(466): JNI WARNING: jarray 0x40590f68 points to non-array object (Ljava/lang/String;)
04-12 01:09:12.557: I/dalvikvm(466): "WebViewCoreThread" prio=5 tid=9 NATIVE
04-12 01:09:12.557: I/dalvikvm(466):   | group="main" sCount=0 dsCount=0 obj=0x40520b58 self=0x116868
04-12 01:09:12.557: I/dalvikvm(466):   | sysTid=475 nice=0 sched=0/0 cgrp=default handle=2708704
04-12 01:09:12.557: I/dalvikvm(466):   | schedstat=( 1932133203 1056054096 197 )
04-12 01:09:12.557: I/dalvikvm(466):   at android.webkit.WebViewCore.nativeTouchUp(Native Method)
04-12 01:09:12.557: I/dalvikvm(466):   at android.webkit.WebViewCore.nativeTouchUp(Native Method)
04-12 01:09:12.587: I/dalvikvm(466):   at android.webkit.WebViewCore.access$3300(WebViewCore.java:53)
04-12 01:09:12.587: I/dalvikvm(466):   at android.webkit.WebViewCore$EventHub$1.handleMessage(WebViewCore.java:1158)
04-12 01:09:12.587: I/dalvikvm(466):   at android.os.Handler.dispatchMessage(Handler.java:99)
04-12 01:09:12.587: I/dalvikvm(466):   at android.os.Looper.loop(Looper.java:123)
04-12 01:09:12.587: I/dalvikvm(466):   at android.webkit.WebViewCore$WebCoreThread.run(WebViewCore.java:629)
04-12 01:09:12.587: I/dalvikvm(466):   at java.lang.Thread.run(Thread.java:1019)
```


http://developer.android.com/guide/webapps/webview.html

Mobile Cross-Platform Development



- WebView component and the possibility to create an interface between Javascript and Android code (or generally to mobile platform native code) is the fundamental element of cross-platform solution available in the market now.

http://phonegap.com/

http://www.appcelerator.com/products/titanium-mobile-application-development/

http://mashable.com/2010/08/11/cross-platform-mobile-development-tools/

Supporting Multiple Screens

- Android runs on a variety of devices that offer different screen sizes and densities.
- The system performs scaling and resizing to make your application work on different screens.
- You should make the effort to optimize your application for different screen sizes and densities. In order to do that you can improve the user experience for all devices and your users believe that your application was actually designed for their devices rather than simply stretched to fit the screen on their devices.

http://developer.android.com/guide/practices/screens_support.html

http://developer.android.com/guide/topics/resources/providing-resources.html#AlternativeResources

Supporting Multiple Screens

- Screen size
  - Actual physical size, measured as the screen's diagonal. Android groups all actual screen sizes into four generalized sizes: **small, normal, large, and extra large**.
- Screen density
  - The quantity of pixels within a physical area of the screen; usually referred to as dpi (dots per inch).
  - Android groups all actual screen densities into four generalized densities: **low, medium, high, and extra high**.
- Orientation
  - The orientation of the screen from the user's point of view.
  - It could be landscape or portrait.
  - Be aware that not only do different devices operate in different orientations by default, but the orientation can change at runtime when the user rotates the device.

Supporting Multiple Screens

- Resolution
  - The total number of physical pixels on a screen.
  - When adding support for multiple screens, applications do not work directly with resolution; applications should be concerned only with screen size and density, as specified by the generalized size and density groups.

- Density-independent pixel (dp)
  - A virtual pixel unit that you should use when defining UI layout, to express layout dimensions or position in a density-independent way.
  - The density-independent pixel is equivalent to one physical pixel on a 160 dpi screen, which is the baseline density assumed by the system for a "medium" density screen.
  - At runtime, the system transparently handles any scaling of the dp units, as necessary, based on the actual density of the screen in use.

How to Support Multiple Screens

- **Explicitly declare in the manifest which screen sizes your application supports**
  - By declaring which screen sizes your application supports, you can ensure that only devices with the screens you support can download your application.
  - Declaring support for different screen sizes can also affect how the system draws your application on larger screens—specifically, whether your application runs in screen compatibility mode.
  - To declare the screen sizes your application supports, you should include the <supports-screens> element in your manifest file.
- **Provide different layouts for different screen sizes**
  - By default, Android resizes your application layout to fit the current device screen.
  - In some cases you might want to change your layout on larger screens for example
    - adjusting the position and size of some elements to take advantage of the additional screen space, or on a smaller screen, you might need to adjust sizes so that everything can fit on the screen.

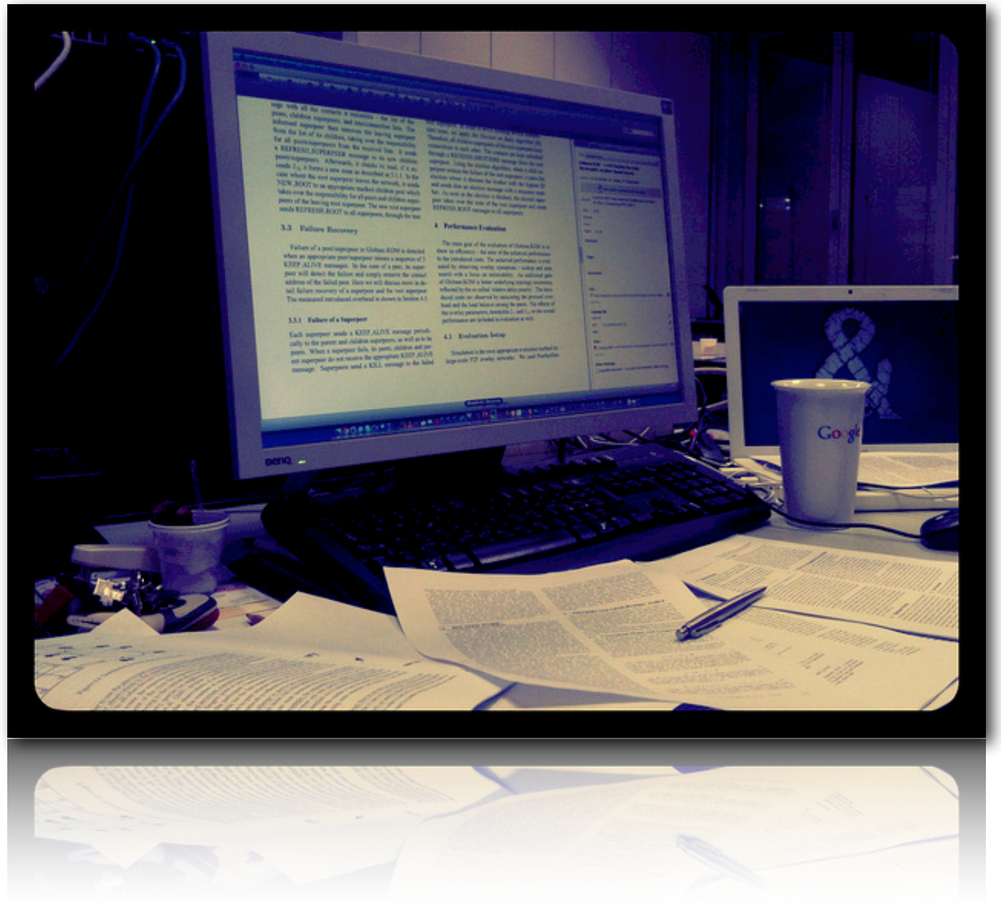
How to Support Multiple Screens

- Provide different bitmap drawables for different screen densities
  - Android scales your bitmap drawables (.png, .jpg, and .gif files) and Nine-Patch drawables (.9.png files) so that they render at the appropriate physical size on each device.
  - If your application provides bitmap drawables only for the baseline, medium screen density (mdpi), then the system scales them up when on a high-density screen, and scales them down when on a low-density screen.
  - The image scaling can cause artifacts in the bitmaps. To ensure your bitmaps look their best, you should include alternative versions at different resolutions for different screen densities.
  - The configuration qualifiers you can use for density-specific resources are: ldpi (low), mdpi (medium), hdpi(high), and xhdpi (extra high).

```
res/layout/my_layout.xml               // layout for normal screen size ("default")
res/layout-small/my_layout.xml         // layout for small screen size
res/layout-large/my_layout.xml         // layout for large screen size
res/layout-xlarge/my_layout.xml        // layout for extra large screen size
res/layout-xlarge-land/my_layout.xml // layout for extra large in landscape orientation

res/drawable-mdpi/my_icon.png          // bitmap for medium density
res/drawable-hdpi/my_icon.png          // bitmap for high density
res/drawable-xhdpi/my_icon.png         // bitmap for extra high density
```

Coming Up

- Next Lecture
  - Location and Mapping
- Homework
  - Review Bookmark Application
  - Implement a custom Toast Notification when a new Bookmark has been saved (Image,Name,Url).
  - Implement a menu to mark a saved Bookmark as “Top Bookmark” and properly show it in the related Tab.



Android Development

Lecture 5
Android Graphical User Interface 3