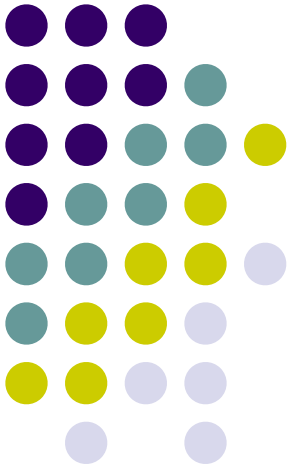
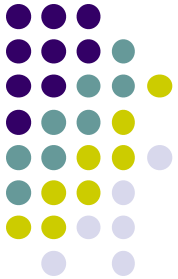
Android Programming

06 – Accessing Internet

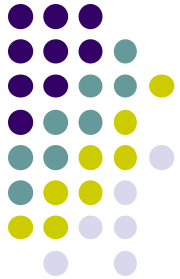


WebView



- This class is the basis upon which you can roll your own web browser or simply display some online content within your Activity
- It uses the WebKit rendering engine to display web pages and includes methods to navigate forward and backward through a history, zoom in and out, perform text searches and more
- To access internet add permission to the manifest
  - <uses-permission android:name=*"android.permission.INTERNET"*/>

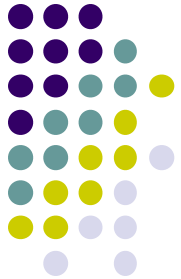
WebView – Basic Usage



- Just create an ACTION_VIEW intent with a URI object and shoot

```
Uri uri = Uri.parse("http://www.example.com");
 Intent intent = new Intent(Intent.ACTION_VIEW, uri);
 startActivity(intent);
```

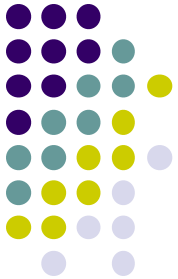
Loading a Web Page



- Invoke loadUrl() method with the url string of the web page

```
@Override
public void onCreate(Bundle icicle) {
}
    super.onCreate(icicle); setContentView(R.layout.main);
    browser=(WebView)findViewById(R.id.webkit);
    browser.loadUrl("http://commonsware.com");
}
```

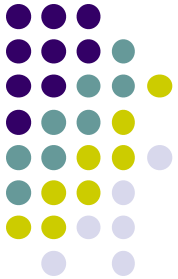
Loading HTML



- Invoke the loadData() method with the required HTML text parameter

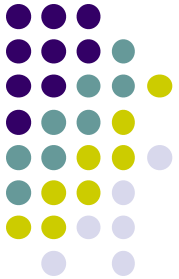
```
@Override
public void onCreate(Bundle icicle) {
}
    super.onCreate(icicle); setContentView(R.layout.main);
    browser=(WebView)findViewById(R.id.webkit);
    browser.loadData("<html<body>Hello, world</body></html>",
            "text/html","UTF-8");
}
```

Accessing Remote Data



- Pre API 22 Apache HTTPClient library was default
- HttpClient library is depracated.
- URL type and HttpURLConnection used instead for better performance.
- To access internet add permission to the manifest
  - <uses-permission android:name=*"android.permission.INTERNET"*/>

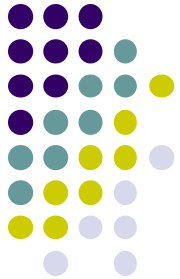
Using the HTTPClient - Deprecated



```
try {
     textView.setText("");
     HttpClient client = new DefaultHttpClient();
     HttpGet request = new HttpGet(urlText.getText().toString());
     HttpResponse response = client.execute(request);
     // Get the response
     BufferedReader rd = new BufferedReader(new InputStreamReader(
                         response.getEntity().getContent()));
     String line = "";
          while ((line = rd.readLine()) != null) {
               textView.append(line);
          }
}

catch (Exception e) {
     System.out.println("Nay, did not work");
     textView.setText(e.getMessage());
}
```

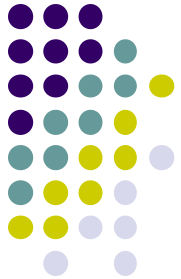
Loading data from Internet takes time, use progress dialogs while loading data as required

Handling HTTP Calls



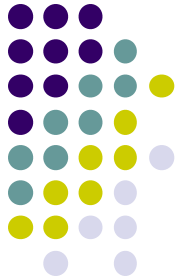
- HttpURLConnection cannot be executed in the main thread
  - NetworkOnMainThreadException thrown
- Use a Thread or AsynchTask for creating and executing request
- Use Handlers to make changes in the UI

How to Use HttpURLConnection



1. Obtain a new HttpURLConnection by calling URL.openConnection() and casting the result to HttpURLConnection.
2. Prepare the request
3. Optionally upload a request body. Instances must be configured with setDoOutput(true) if they include a request body. Transmit data by writing to the stream returned by getOutputStream().
4. Read the response (getInputStream())
5. Disconnect

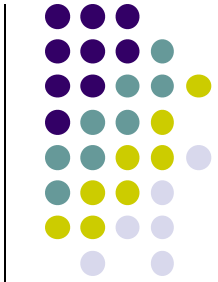
HttpURLConnection Example



```
URL url = new URL("http://www.android.com/");
   HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
   try {
     InputStream in = new BufferedInputStream(urlConnection.getInputStream());
     readStream(in);
    finally {
     urlConnection.disconnect();
   }
 }
```

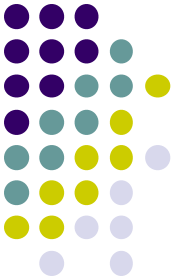
For reading the streams BufferedReader or other reader types can be used.

Posting Content



- To upload data to a web server, configure the connection for output using setDoOutput(true).
- If content length known: setFixedLengthStreamingMode(int)
- If content length not known: setChunkedStreamingMode(int)
  - *“urlConnection.setRequestHeader("Transfer-Encoding","chunked");” call must also be added*

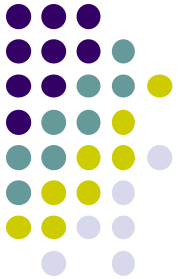
Posting Example



```
HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
   try {
     urlConnection.setDoOutput(true);

     urlConnection.setRequestMethod("POST");
     urlConnection.setRequestProperty("Content-Type",
                              "application/x-www-form-urlencoded");

     String params =“name=somename&lastname=somelastname”;
     urlConnection.setFixedLengthStreamingMode(params.getBytes().length);

     OutputStream out = new BufferedOutputStream(urlConnection.getOutputStream());
     out.write(params);
     out.flush();
     out.close();
   finally {
     urlConnection.disconnect();
   }
 }
```

HttpURLConnection uses the GET method by default.
It will use POST if setDoOutput(true) has been called.
Other HTTP methods (OPTIONS, HEAD, PUT, DELETE and TRACE) can be used with setRequestMethod(String).

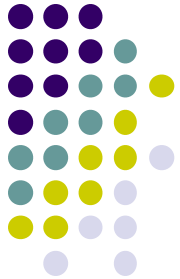
Managing Session



- The default CookieManager keeps all accepted cookies in memory. It will forget these cookies when the VM exits
- Enable VM-wide cookie management using CookieHandler and CookieManager:

```
CookieManager cookieManager = new CookieManager();
  CookieHandler.setDefault(cookieManager);
```

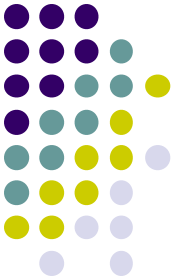
Avoiding Bugs in Early Releases



- Prior to Android 2.2 (Froyo), this class had some frustrating bugs. In particular, calling close() on a readable InputStream could poison the connection pool. Work around this by disabling connection pooling:

```
private void disableConnectionReuseIfNecessary() {
   // Work around pre-Froyo bugs in HTTP connection reuse.
   if (Integer.parseInt(Build.VERSION.SDK) < Build.VERSION_CODES.FROYO) {
     System.setProperty("http.keepAlive", "false");

 }}
```

Parse Json Data

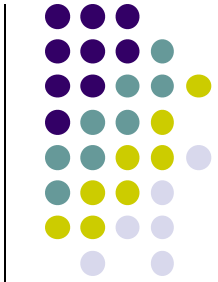


- Getting data from a Json web service requires an HTTPclient and same operations in HTTP access
- Then we use the built in Json objects to parse data
- If Json data is mapped with object names we use a JsonObject, else we use JsonArray

String containing JsonData

```
JSONArray arr = new JSONArray(result);
    for (int i = 0; i < arr.length(); i++) {
    JSONObject current = arr.getJSONObject(i);
    Director d = new Director();
    d.setName(current.getString("name"));
    d.setLastname(current.getString("lastname"));
    d.setId(current.getInt("id"));
    directors.add(d);

}
```

Best Practices



- Soap transfers more data and requires more processing than JSON
- Try using restful web services and JSON for data exchange