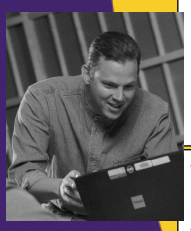
# Network Programming: Part II (HTTP-Specific Techniques)

Originals of Slides and Source Code for Examples:
http://www.coreservlets.com/android-tutorial/



© 2011 Marty Hall


## For live Android training, please see courses at http://courses.coreservlets.com/.



**Taught by the author of *Core Servlets and JSP*, *More Servlets and JSP*, and this Android tutorial. Available at public venues, or customized versions can be held on-site at your organization.**

- Courses developed and taught by Marty Hall
  - Android development, JSF 2, servlets/JSP, Ajax, jQuery, Java 6 programming, custom mix of topics
  - Ajax courses can concentrate on 1 library (jQuery, Prototype/Scriptaculous, Ext-JS, Dojo, etc.) or survey several
- Courses developed and taught by coreservlets.com experts (edited by Marty)
  - Spring, Hibernate/JPA, EJB3, GWT, RESTful and SOAP-based Web Services

**Contact hall@coreservlets.com for details**

# Topics in This Section

- **Part I (previous section): general networking**
  - Socket basics
  - Requesting Internet permission
  - Example: NIST atomic time
  - Aside: simple String formatting and parsing
  - Example: FTP welcome messages
  - Example: validating URLs with HEAD
- **Part II (this section): HTTP-specific approaches**
  - HttpURLConnection
  - HttpClient
  - Examples: Searching Web pages
  - Using JSON
  - Example: remote loan calculations
  - Example: Google translation services





# Overview

# Big Idea

- **Many ways to communicate with a server**
  - Socket class
    - Lets you do general-purpose network programming
      - Same as with desktop Java programming
  - HttpURLConnection
    - Simplifies connections to HTTP servers
      - Same as with desktop Java programming
  - HttpClient
    - Simplest way to download entire content of a URL
      - Not standard in Java SE, but standard in Android
  - JSONObject
    - Simplifies creation and parsing of JSON data
      - Not standard in Java SE, but standard in Android





# HttpURLConnection

# URL and HttpURLConnection: Overview

- **URL**
  - The URL class can parse a URL and extract its components (protocol, host, port, URI, etc.)
  - You can use openConnection to get a stream to the URL
- **HttpURLConnection**
  - If the URL's protocol is HTTP, cast the result of openConnection to HttpURLConnection
    - Lets you read the status code
      - 200, 301, 404, etc.
    - Lets you set request headers
      - Accept, Referer, User-Agent, etc.
    - Lets you read response headers
      - Content-Type, Location, etc.
    - Special case for cookies
      - Use CookieManager instead of raw headers



# URL: Details

- **openConnection**
  - Establishes connection to server.
    - Cast result to HttpURLConnection for more control
    - Uses GET by default. To use POST, call setDoOutput(true) and send data via openOutputStream
- **openInputStream**
  - Gets a Stream for reading data. Wrap in InputStreamReader (usually buffered) to use readLine.
- **getProtocol, getHost, getPort, getFile, getRef**
  - Returns the different components of the URL.
    - The port is -1 if none specified, so you can distinguish an omitted port from an explicit 80.

# Simple URL Methods: Example

```
import java.net.*;

public class UrlTest {
  public static void main(String[] args) {
    if (args.length == 1) {
      try {
        URL url = new URL(args[0]);
        System.out.println
          ("URL: " + url.toExternalForm() + "\n" +
           "  File:      " + url.getFile() + "\n" +
           "  Host:      " + url.getHost() + "\n" +
           "  Port:      " + url.getPort() + "\n" +
           "  Protocol:  " + url.getProtocol() + "\n" +
           "  Reference: " + url.getRef());
      } catch(MalformedURLException mue) {
        System.out.println("Bad URL.");
      }
    } else
      System.out.println("Usage: UrlTest <URL>");
  }
}
```



# Simple URL Methods: Results

```
> java UrlTest http://www.irs.gov/mission/#squeezing-them-dry
URL: http://www.irs.gov/mission/#squeezing-them-dry
  File:      /mission/
  Host:      www.irs.gov
  Port:      -1
  Protocol:  http
  Reference: squeezing-them-dry
```

What month do you think it was when I first wrote this example?

# HttpURLConnection: Details

- **Getting a connection from a URL**

```
URL url = new URL("http://…");
HttpURLConnection urlConnection =
   (HttpURLConnection)url.openConnection();
```

- **Reading data**

```
BufferedReader in =
   new BufferedReader(new InputStreamReader
                                (urlConnection.getInputStream()));
while ((line = in.readLine()) != null) {
   doSomethingWith(line);
}
```

- **Other methods**
  - disconnect, getResponseCode, getHeaderField
    - Call disconnect when done





# Example: Searching Web Pages for Keywords (Version 1)

# URL Searcher

- **Idea**
  - Print out all lines in a given URL that contain keyword
- **Approach**
  - Read URL and keyword from user
  - Call theUrl.openConnection and cast result to HttpURLConnection
  - Attach BufferedReader to the HttpURLConnection
  - Read one line at a time (until null), checking if the line contains the keyword
  - Catch MalformedURLException and IOException
  - Call disconnect() when done



# Manifest File (AndroidManifest.xml)

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.coreservlets.networking"
      android:versionCode="1"
      android:versionName="1.0">
    <uses-sdk android:minSdkVersion="8" />
    <uses-permission android:name="android.permission.INTERNET" />
    ...
</manifest>
```

# Layout File (res/layout/url_searcher.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://..."
              android:orientation="vertical" ...>
    <LinearLayout ...>
        <TextView android:text="@string/url_prompt" .../>
        <EditText android:id="@+id/url_to_search" ... > ...
        </EditText>
    </LinearLayout>
    <LinearLayout ... >
        <TextView android:text="@string/search_string_prompt" ... />
        <EditText android:id="@+id/search_string" ...></EditText>
    </LinearLayout>
    <Button android:text="@string/url_checker_button_text" ...
            android:onClick="searchInUrl"/>
    <ScrollView  ...>
        <TextView android:id="@+id/url_search_result" ... />
    </ScrollView>
</LinearLayout>
```



# Values Files

- **res/values/strings.xml**
  - Defines title, prompts, and button labels for all Activities in this section.
    - Used in both portrait and landscape mode.
- **res/values/colors.xml**
  - Defines foreground color for some of the results
    - Used in both portrait and landscape mode.
- **res/values/dimens.xml**
  - Gives font sizes.
    - Used in portrait mode.
- **res/values-land/dimens.xml**
  - Gives font sizes.
    - Used in landscape mode.

# Main Activity (UrlSearcher1Activity.java)

```
public class UrlSearcher1Activity extends Activity {
    protected EditText mUrlToSearch, mSearchString;
    protected TextView mUrlMessageResult;
    protected float mResultTextSize, mErrorTextSize;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.url_searcher);
        mUrlToSearch = (EditText)findViewById(R.id.url_to_search);
        mSearchString = (EditText)findViewById(R.id.search_string);
        mUrlMessageResult =
               (TextView)findViewById(R.id.url_search_result);
        Resources resources = getResources();
        mResultTextSize =
                resources.getDimension(R.dimen.url_search_results_size);
        mErrorTextSize =
                resources.getDimension(R.dimen.url_search_error_size);
    }
```

Note the protected fields and (next slides) methods – next example will extend this class and share much of the code.



# Main Activity, Continued (UrlSearcher1Activity.java)

```
    public void searchInUrl(View clickedButton) {
        String urlString =
                mUrlToSearch.getText().toString();
        String searchString =
                mSearchString.getText().toString();
        showResults(urlString, searchString);
    }
```

This is the method called by the Button. Next example will inherit it unchanged.

# Main Activity, Continued (UrlSearcher1Activity.java)

```
protected void showResults(String urlString, String searchString) {
    HttpURLConnection urlConnection = null;
    try {
        URL url = new URL(urlString);
        urlConnection = (HttpURLConnection)url.openConnection();
        BufferedReader in =
                new BufferedReader
                    (new InputStreamReader(urlConnection.getInputStream()));
        String line;
        StringBuilder matches = new StringBuilder("");
        int lineNum = 0;
        int numMatches = 0;
        while ((line = in.readLine()) != null) {
            lineNum++;
            if(line.contains(searchString)) {
                numMatches++;
                matches.append(makeMatch(line, lineNum));
            }
        }
        displayResults(matches, numMatches);
```

This is the method that is overridden in the next example.



# Main Activity, Continued (UrlSearcher1Activity.java)

```
    } catch (MalformedURLException mue) {
        showError("Bad URL: " + urlString);
        mue.printStackTrace(); // View this in DDMS window
    } catch (IOException ioe) {
        showError("Error in connection: " + ioe);
        ioe.printStackTrace(); // View this in DDMS window
    } finally {
        if (urlConnection != null) {
            urlConnection.disconnect();
        }
    }
}
```

This is a continuation of the showResults method started on previous slide.

# Main Activity, Continued (UrlSearcher1Activity.java)

```
    protected String makeMatch(String text, int lineNum) {
        return(String.format("LINE %s: %s%n", lineNum, text));
    }

    protected void displayResults(StringBuilder matches,
                                  int numMatches) {
        if (numMatches > 0) {
            showMatches(matches, numMatches);
        } else {
            showError("No matches");
        }
    }
```



# Main Activity, Continued (UrlSearcher1Activity.java)

```
    protected void showMatches(StringBuilder matches,
                               int numMatches) {
        String introMessage =
                String.format("FOUND %s MATCHES:%n%n",
                              numMatches);
        matches.insert(0, introMessage);
        mUrlMessageResult.setTextSize(mResultTextSize);
        mUrlMessageResult.setText(matches);
    }

    protected void showError(String text) {
        mUrlMessageResult.setTextSize(mErrorTextSize);
        mUrlMessageResult.setText("\n\n" + text);
    }
}
```

# Results






# HttpClient

# HttpClient

- **Idea**
  - A popular class from the Apache HttpComponents suite that is bundled with Android.
    - Minor problem: Android does not document precisely which version it uses, so hard to use Apache tutorials
- **Capabilities**
  - Simple way to read an entire URL (via GET) into String
  - Moderately simple way to send POST parameters, then read entire result into a String
  - Many, many other capabilities
    - But only the simple reading of content is shown in this tutorial. For other capabilities, see http://hc.apache.org/httpclient-3.x/tutorial.html



# HttpClient: Reading Result of GET Request

- **Make a default client**
  - HttpClient client = new DefaultHttpClient();
- **Make an HttpGet with address**
  - HttpGet httpGet = new HttpGet(address);
- **Make a String ResponseHandler**
  - ResponseHandler<String> handler =
    new BasicResponseHandler();
- **Call client.execute**
  - String content = client.execute(httpGet, handler);

# HttpClient: Reading Result of POST Request

- **Make a default client**

```
HttpClient client = new DefaultHttpClient();
```

- **Make an HttpPost with address**

```
HttpPost httpPost = new HttpPost(address);
```

- **Make a List of name/value pairs**

```
List<NameValuePair> params =  new ArrayList<NameValuePair>();
params.add(new BasicNameValuePair(paramName1, paramValue1));
params.add(…);  // More names and values. NOT URL-encoded
```

- **Attach POST data**

```
UrlEncodedFormEntity entity = new UrlEncodedFormEntity(params, "UTF-8");
httpPost.setEntity(entity);
```

- **Make a String ResponseHandler**

```
ResponseHandler<String> handler = new BasicResponseHandler();
```

- **Call client.execute**

```
String content = client.execute(httpPost, handler);
```



# HttpUtils

- **Idea**
  - A class developed for this tutorial that wraps up some simple HttpClient functionality
- **Static methods**
  - urlContent(String address)
    - Returns a String containing entire content of address
    - Uses GET. If you want query data, URL encode it and attach it to the end of URL after question mark
  - urlContentPost(String address, String ... paramNamesAndValues)
    - Returns a String containing entire content of address
    - Uses POST. All args after the first are alternating parameter names and unencoded parameter values.

# HttpUtils

```
public static String urlContent(String address)
    throws IOException, ClientProtocolException {
  HttpClient client = new DefaultHttpClient();
  HttpGet httpGet = new HttpGet(address);
  ResponseHandler<String> handler =
    new BasicResponseHandler();
  return(client.execute(httpGet, handler));
}
```



# HttpUtils, Continued

```
public static String urlContentPost(String address,
                                    String ... paramNamesAndValues)
    throws IOException, ClientProtocolException {
  HttpClient client = new DefaultHttpClient();
  HttpPost httpPost = new HttpPost(address);
  List<NameValuePair> params = new ArrayList<NameValuePair>();
  for(int i=0; i<paramNamesAndValues.length-1; i=i+2) {
    String paramName = paramNamesAndValues[i];
    String paramValue = paramNamesAndValues[i+1];  // NOT URL-Encoded
    params.add(new BasicNameValuePair(paramName, paramValue));
  }
  UrlEncodedFormEntity entity = new UrlEncodedFormEntity(params, "UTF-8");
  httpPost.setEntity(entity);
  ResponseHandler<String> handler = new BasicResponseHandler();
  return(client.execute(httpPost, handler));
}
```

The POST version is a bit more complicated, but GET is usually what you want. Full code for HttpUtils (as with all code in the tutorial) can be downloaded from tutorial home page.

# Example: Searching Web Pages for Keywords (Version 2)



## URL Searcher

- **Idea**
  - Redo the previous example, but read URL content with HttpClient instead of HttpURLConnection
- **Approach**
  - Read URL and keyword from user
  - Call HttpUtils.urlContent(address) to get entire content of address as a single String
  - Use String.split to split String on CR or LF
  - Test each array entry to see if it contains keyword
  - Reuse most of the code from the previous example

# XML Files

- **AndroidManifest.xml**
  - Requests Internet permission as shown previously. Also declares the new Activity.
- **res/layout/url_searcher.xml**
  - Reuses previous layout file with *no* changes
- **Values files**
  - Reuses previous files with *no* changes
    - strings.xml
    - colors.xml
    - dimens.xml



# Main Activity (UrlSearcher2Activity.java)

```
public class UrlSearcher2Activity extends UrlSearcher1Activity {
    @Override
    protected void showResults(String urlString,
                               String searchString) {
        try {
            String urlBody = HttpUtils.urlContent(urlString);
            String[] lines = urlBody.split("[\\n\\r]+");
            StringBuilder matches = new StringBuilder("");
            int lineNum = 0;
            int numMatches = 0;
            for (String line: lines) {
                lineNum++;
                if(line.contains(searchString)) {
                    numMatches++;
                    matches.append(makeMatch(line, lineNum));
                }
            }
            displayResults(matches, numMatches);
```

All methods except showResults are inherited from previous class.

# Main Activity, Continued (UrlSearcher2Activity.java)

```
        } catch (ClientProtocolException cpe) {
            showError("Bad URL: " + urlString);
            cpe.printStackTrace(); // View this in DDMS window
        } catch (IOException ioe) {
            showError("Error in connection: " + ioe);
            ioe.printStackTrace(); // View this in DDMS window
        }
    }
}
```



# Results


Identical to previous example.

# Aside: JSON Format

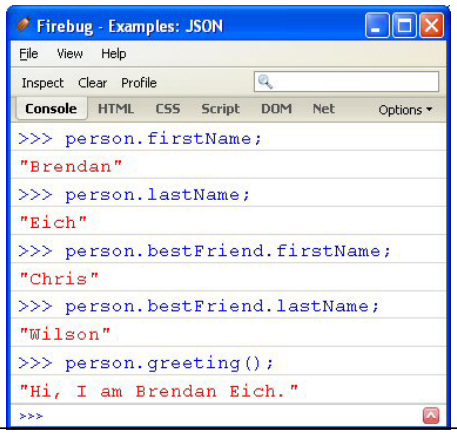


# JSON (JavaScript Object Notation)

- **Idea**
  - A simple textual representation of (JavaScript) objects
    - Called "object literals" or "anonymous objects"
  - Main applications in JavaScript
    - One-time-use objects (rather than reusable classes)
    - Objects received via strings
- **Directly in JavaScript**
  - `var someObject = { property1: value1, property2: value2, ... };`
- **Widely used in other languages**
  - Lightweight alternative to XML
  - Android has a builtin class (JSONObject) that will both build and parse strings representing JSON

# JSON: Example in JavaScript

```
var person =
  { firstName:  'Brendan',
    lastName:   'Eich',
    bestFriend: { firstName: 'Chris',
                  lastName:  'Wilson' },
    greeting: function() {
                  return("Hi, I am " + this.firstName +
                         " " + this.lastName + ".");
                }
  };
```






# Strict JSON

- **Object literals in JavaScript**
  - Any way you would type a data structure into JavaScript.
    - { firstName: 'Larry', lastName: 'Page'}
- **Strict JSON according to json.org**
  - Subset of JavaScript where:
    - Object property names must be in double quotes
    - Strings are represented with double quotes only (not single quotes)
      - { "firstName": "Larry", "lastName": "Page"}
  - This is what Android's JSONObject supports
- **MIME type for JSON from server**
  - RFC 4627 says JSON should have "application/json" type
  - No known libraries enforce this

# Popular Web Services that Send JSON Data

- **Google APIs**
  - Search, translate, data, and many more
    - http://code.google.com/apis/customsearch/v1/overview.html
- **Yahoo APIs**
  - Search, travel, answers
    - http://developer.yahoo.com/
- **Twitter APIs**
  - https://dev.twitter.com/
- **GeoNames**
  - http://www.geonames.org/export/web-services.html
- **Flickr**
  - http://www.flickr.com/services/api/
- **Thousands of others**
  - See list here
    - http://www.programmableweb.com/apis/directory/1?format=JSON





# The JSONObject Class

# JSONObject

- **Idea**
  - A popular JSON-in-Java library from json.org that is bundled with Android.
    - Major problem: Android does not document precisely which version it uses, and the version it does use is long-obsolete and lacks JSON-from-bean capability
- **Capabilities**
  - Builds a JSONObject
    - From String, Map, or bean
  - Lets you extract data from JSONObject
    - getString(name), getDouble(name), etc
  - Lets you output string representing a JSONObject
    - toString



# Building a JSONObject

- **Constructors**
  - new JSONObject(jsonString)
    - Reads a JSON string and builds JSONObject from it.
      - This is most important constructor for use directly on Android, since main Android use of JSON is handling JSON from Web services
  - new JSONObject(nameValueMap)
    - Takes a Map and builds JSON from it. Used on Android if you want to send JSON to a Web service
  - new JSONObject(bean) [not in Android version!]
    - Takes a bean and builds JSON based on bean properties. Cannot use directly on Android, but very useful on server that sends JSON to Android
      - If Android had supported this version, would have been much better way to send JSON to a Web service

# Extracting Data From a JSONObject

- **Accessors**
  - get(propertyName)
    - Returns Object associated with name
  - getString(propertyName)
    - Returns String associated with name. Works for any type (if Object, uses its toString method)
  - getDouble(propertyName)
    - Returns double associated with name. Throws JSONException if value cannot be converted to double.
  - get*Blah*(propertyName)
    - getInt, getBoolean, etc. Similar in spirit to getDouble.
  - getJSONArray(propertyName)
    - Returns JSONArray (not native Java array!) associated with name



# Other Capabilities

- **toString**
  - Builds strict-JSON string from JSONObject
- **The JSONArray class**
  - Represents an array of JSON.
  - Although it can be used to send data from Android, it is usually used for received data.
    - { "names": ["larry", "curly", "moe"],
      "primes": [2, 3, 5, 7, 11] }

# Using JSONObject on Server



# Steps for Servlet Outputting JSON (for Android Use)

- **Usual tasks**
  - response.setContentType("application/json");
  - PrintWriter out = response.getWriter
  - Call normal business logic code to get bean, array, or Map
- **New tasks**
  - Turn Java object into JSONObject
    - JSONObject result = new JSONObject(bean);
    - JSONArray result = new JSONArray(arrayOfBeans, false);
    - JSONObject result = new JSONObject(map);
  - Output JSONObject with print
    - out.print(result);
      - Automatically calls toString, which does the real JSONification

# JSONObject's Algorithm for JSONifying a Java Object

- **Find the bean properties**
  - E.g., Name class has getFirstName, and getLastName methods, so properties are firstName and lastName
- **Call the associated accessor methods**
  - Call each of getFirstName and getLastName on object
- **Build JSON representation**
  - JavaScript property names are bean property names
  - Values are results of getter methods
- **Mini Example**
  - toString of a JSONObject representing a Name
    - Java: make new Name("John", "Resig").
      Pass to JSONObject, then call print (which uses toString)
    - Result: { "firstName": "John", "lastName": "Resig"}



# Finding Bean Property Name Based on Method Name

- **Usual rule**
  - Drop the word "get" and change the next letter to lowercase. Instance variable name is irrelevant.
    - Method name: getUserFirstName
    - Property name: userFirstName
- **Exception 1: boolean properties**
  - If getter returns boolean or Boolean
    - Method name: getPrime or isPrime
    - Property name: prime
- **Exception 2: consecutive uppercase letters**
  - If two uppercase letters in a row after "get"
    - Method name: getURL
    - Property name: URL (not uRL)

# Bean Properties: Examples

| Method Name | Property Name | Example JSON Output |
|---|---|---|
| getFirstName | firstName | { "firstName": "Joe", … } |
| isExecutive (boolean property) | executive | { "executive": true, … } |
| getExecutive (boolean property) | executive | { "executive": true, … } |
| getZIP | ZIP | { "ZIP": "12345", … } |

Note 1: property name does not exist anywhere in your code. It is just a shortcut for the method name.
Note 2: property name is derived only from method name. Instance variable name is irrelevant.



# Steps for Using JSON Utilities: Sample Servlet Code

```
public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
    throws ServletException, IOException {
  response.setContentType("application/json");
  PrintWriter out = response.getWriter();
  SomeBean javaResult = callSomeBusinessLogic(…);
  JSONObject jsonResult =
    new JSONObject(javaResult);
  out.print(jsonResult);
}
```

Lines in red are the only ones that typically change from application to application. Other lines stay exactly as is.

Calling print automatically calls toString behind the scenes

# Drawbacks of JSONification

- **Sends state, not behavior, of object**
  - So, properties that are derived from other properties will be inconsistent on client if other properties are changed
- **Example: Java "Circle" class**
  - getRadius
    - Returns instance variable
  - getArea
    - Returns Math.PI*radius*radius
- **JSONified instance**
  - Example
    - Make new Circle(10). Pass to JSONObject, call toString
    - Result: { "radius": 10.0, "area": 314.15927…}
  - Problem
    - If client-side code changes radius, area doesn't match



# JSONObject from Bean: Example Code

```
public class CityTest1 {
  public static void main(String[] args) {
    City sf = CityUtils.getCity("San Francisco");
    JSONObject sfJSON = new JSONObject(sf);
    System.out.println("JSON version of SF is:\n" +
                       sfJSON);
  }
}
```

City is a simple class with a bunch of getter methods (getTime, getName, getPopulation, etc.)

Note: toString is automatically called when you print an Object in Java. It is the toString method of JSONObject that builds the JSON representation.

# JSONObject from Bean: Example Result

```
JSON version of SF is:
{"time":       "06:00:55 AM",
 "name":       "San Francisco",
 "timeZone":   -3,
 "pop":        744041,
 "population": "  744,041"}
```

*(White space added for readability)*





# Example: Remote Loan Calculator

# JSON-Based Loan Calculator Service

- **Idea**
  - Make servlet for doing loan calculations
    - Input: JSON object with amount, period, and interest rate
    - Output: JSON object with inputs plus monthly payment and total payments
- **Approach**
  - Server
    - Read parameter and pass to JSONObject constructor
    - Extract values with getDouble and getLong
    - Use same PaymentInfo class as in several previous examples
    - Turn PaymentInfo into JSON via JSONObject
  - Client (Android)
    - Build Map and turn into JSONObject, send to server
    - Read String, turn into JSONObject
    - Extract info with getString



# Servlet

```
@WebServlet("/loan-calculator")
public class LoanCalculator extends HttpServlet {
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        String inputString = request.getParameter("loanInputs");
        double loanAmount = 200000;
        double annualInterestRateInPercent = 5.5;
        long loanPeriodInMonths = 360;
        try {
            JSONObject inputValues = new JSONObject(inputString);
            loanAmount = inputValues.getDouble("amount");
            annualInterestRateInPercent = inputValues.getDouble("rate");
            loanPeriodInMonths = inputValues.getLong("months");
        } catch (Exception e) {  // NullPointerException & JSONException
            // Use default values assigned before the try block
        }
        PaymentInfo info = new PaymentInfo(loanAmount,
                                           annualInterestRateInPercent,
                                           loanPeriodInMonths);
        PrintWriter out = response.getWriter();
        out.println(new JSONObject(info));
    }
```

If you are unfamiliar with servlets, see extensive tutorial series at http://www.coreservlets.com/

# Servlet, Continued

```
@Override
public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
        throws ServletException, IOException {
    doGet(request, response);
}
```

Makes the most sense to send the data via POST. However, it also supports GET to simplify interactive testing by editing the browser line.

This app is installed on apps.coreservlets.com, so URL is http://apps.coreservlets.com/NetworkingSupport/loan-calculator
You can experiment interactively by calling
http://apps.coreservlets.com/NetworkingSupport/loan-calculator?loanInputs={amount: 200000, rate: 6.5, months: 180}

The tutorial Web site has the servlet code as well as the Android code, so you can install the servlet on a local Java server (requires Tomcat 7 or equivalent). However, note that the Android emulator does not understand localhost as a host name, since it acts like independent device. So, you must use real domain name or IP address.



# Android Activity

```
public class LoanCalculatorActivity extends Activity {
    private EditText mBaseUrl, mLoanAmount, mInterestRate, mLoanPeriod;
    private TextView mMontlyPaymentResult, mTotalPaymentsResult;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.loan_calculator);
        mBaseUrl = (EditText)findViewById(R.id.base_url);
        mLoanAmount = (EditText)findViewById(R.id.loan_amount);
        mInterestRate = (EditText)findViewById(R.id.interest_rate);
        mLoanPeriod = (EditText)findViewById(R.id.loan_period);
        mMontlyPaymentResult =
                (TextView)findViewById(R.id.monthly_payment_result);
        mTotalPaymentsResult =
                (TextView)findViewById(R.id.total_payments_result);
    }
```

# Android Activity, Continued

```
public void showLoanPayments(View clickedButton) {
    String baseUrl = mBaseUrl.getText().toString();
    String loanAmount = mLoanAmount.getText().toString();
    String interestRate = mInterestRate.getText().toString();
    String loanPeriod = mLoanPeriod.getText().toString();
    LoanInputs inputs =
        new LoanInputs(loanAmount, interestRate, loanPeriod);
    JSONObject inputsJson = new JSONObject(inputs.getInputMap());
    try {
        String jsonString =
            HttpUtils.urlContentPost(baseUrl, "loanInputs",
                                     inputsJson.toString());
        JSONObject jsonResult = new JSONObject(jsonString);
        mMontlyPaymentResult.setText
            (jsonResult.getString("formattedMonthlyPayment"));
        mTotalPaymentsResult.setText
            (jsonResult.getString("formattedTotalPayments"));
        mLoanAmount.setText
            (jsonResult.getString("loanAmount"));
        mInterestRate.setText
            (jsonResult.getString("annualInterestRateInPercent"));
        mLoanPeriod.setText
            (jsonResult.getString("loanPeriodInMonths"));
```



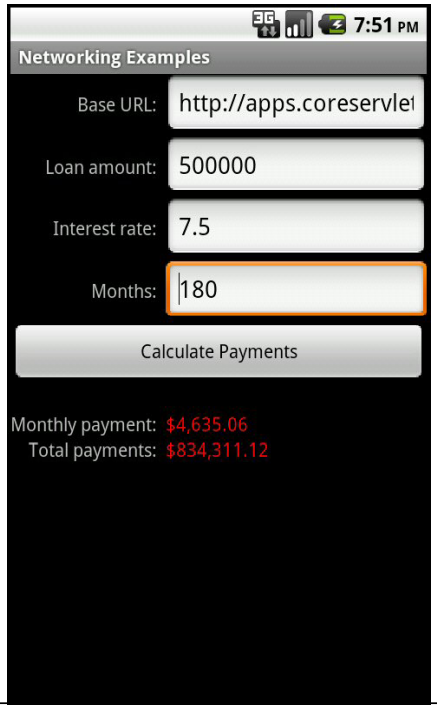
# Helper Class

```
public class LoanInputs {
    private Map<String,String> mInputMap;

    public LoanInputs(String amount, String rate,
                      String months) {
        mInputMap = new HashMap<String,String>();
        mInputMap.put("amount", amount);
        mInputMap.put("rate", rate);
        mInputMap.put("months", months);
    }

    public Map<String,String> getInputMap() {
        return(mInputMap);
    }
}
```

# Results




http://apps.coreservlets.com/NetworkingSupport/loan-calculator





# Example: Google Translation Services

# Double Translate App

- **Idea**
  - The Google Translate services has a JSON-based API
    - http://code.google.com/apis/language/translate/v2/using_rest.html
    - Can look up all the supported languages
    - Can translate from any supported language into any other
  - Translate from English to foreign and then back to English to see how well the translation works
    - Very good for literal text with simple grammar.
- **Approach**
  - Get JSONArray representing supported lanuages.
    - Put in Spinner
  - Get JSONObject representing intermediate translation
    - Send back to server to get final translation
  - Only selected code will be shown here. Full code online.



# Main Activity

```
public class DoubleTranslateActivity extends Activity {
    private EditText mText;
    private Spinner mLanguageChoices;
    private TextView mIntermediateResult, mFinalResult;
    private String apiKey = "...";
```

The translate API requires you to apply for a free API key and send the key with every request.

# Main Activity, Continued

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    mText = (EditText)findViewById(R.id.text);
    mLanguageChoices = (Spinner)findViewById(R.id.language_choices);
    List<Language> supportedLanguages =
        TranslateUtils.supportedLanguages(apiKey);
    ArrayAdapter<Language> spinner2Adapter =
        new ArrayAdapter<Language>(this,
                                   android.R.layout.simple_spinner_item,
                                   supportedLanguages);
    spinner2Adapter.setDropDownViewResource
       (android.R.layout.simple_spinner_dropdown_item);
    mLanguageChoices.setAdapter(spinner2Adapter);
    mIntermediateResult = (TextView)findViewById(R.id.intermediate_result);
    mFinalResult = (TextView)findViewById(R.id.final_result);
}
```



# Main Activity, Continued

```
public void doubleTranslate(View clickedButton) {
    String sourceLanguageCode = "en";
    Language targetLanguage =
        (Language)mLanguageChoices.getSelectedItem();
    String targetLanguageCode = targetLanguage.getCode();
    String textToTranslate = mText.getText().toString();
    Translator translator =
        new Translator(apiKey, sourceLanguageCode,
                       targetLanguageCode, textToTranslate);
    String firstTranslation = translator.getFirstTranslation();
    mIntermediateResult.setText(firstTranslation);
    String secondTranslation =
        translator.getSecondTranslation();
    mFinalResult.setText(secondTranslation);
}
```

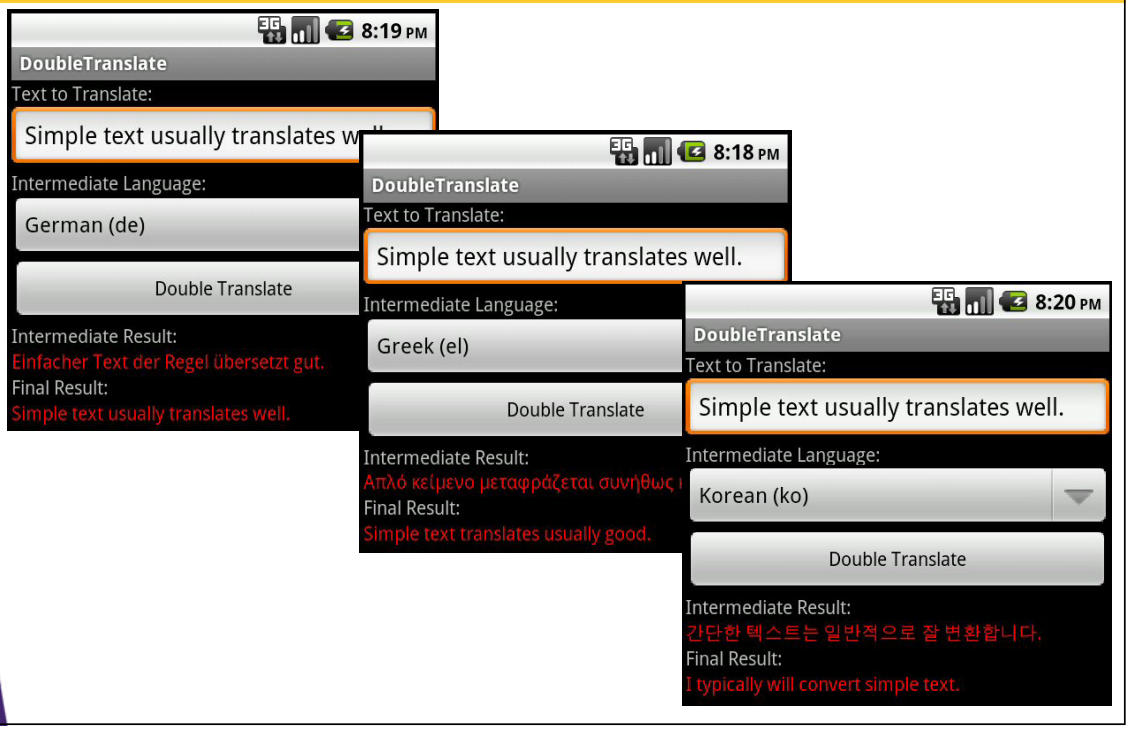
# Key Helper Class (TranslationUtils)

```
public static String translationString
            (String apiKey, String sourceLanguage,
             String targetLanguage, String textToTranslate) {
  try {
    String jsonString = translationJson(apiKey, sourceLanguage,
                                        targetLanguage, textToTranslate);
    JSONObject jsonObject = new JSONObject(jsonString);
    String text =
      jsonObject.getJSONObject("data").getJSONArray("translations")
                .getJSONObject(0).getString("translatedText");
    return(text);
  } catch(HttpResponseException hre) {
    int errorCode = hre.getStatusCode();
    if (errorCode == 403) { // 403 is "Forbidden"
      return(ERROR_FORBIDDEN);
    } else { // Probably 400, "Bad Request"
      return(ERROR_BAD_REQUEST);
    }
  } catch(IOException ioe) {
    return(ERROR_BAD_CONNECTION);
  } catch(JSONException jse) {
    return(ERROR_JSON_PROBLEM);
  }
}
```



# Results: Very Simple Text

# Results: Medium-Difficulty Text

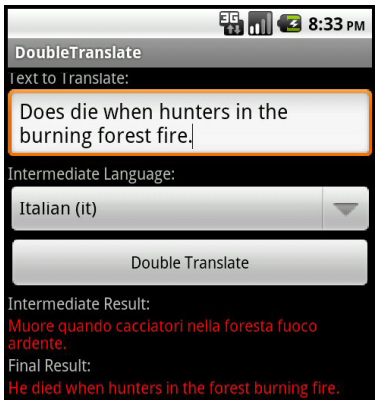


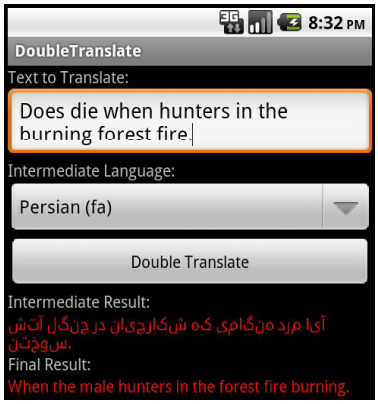


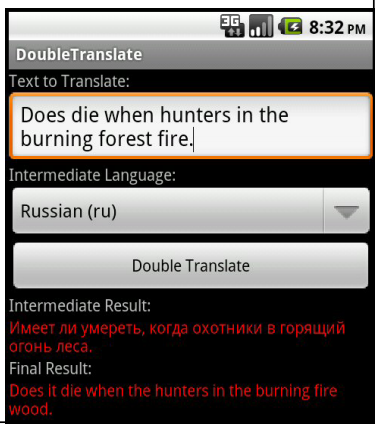

# Results: Deliberately-Tricky Text


The proper interpretation is “old people crew (work on) the boat”.
This is an example of a garden-path sentence and is taken from
http://en.wikipedia.org/wiki/Garden_path_sentence

# Results: More Deliberately-Tricky Text




The proper interpretation is "baby deer die when hunters (who are in the burning forest) shoot at them".





# Wrap-Up

# More Reading

- **JavaDoc**
  - HttpURLConnection
    - http://developer.android.com/reference/java/net/HttpURLConnection.html
- **HttpClient Tutorial**
  - http://hc.apache.org/httpclient-3.x/tutorial.html
- **JSON Tutorials**
  - http://json.org/
  - http://www.webmonkey.com/2010/02/get_started_with_json/
  - http://www.hunlock.com/blogs/Mastering_JSON_(_JavaScript_Object_Notation_)



# Summary

- **Read individual lines from a URL**
  - Use HttpURLConnection
- **Read an entire URL as a String**
  - Use HttpClient
    - Particularly useful if result is JSON
- **JSON**
  - Simple format widely used in Ajax and Web services
- **JSONObject**
  - Converts string into JSON
  - Outputs JSON string
    - But builtin Android version lacks most important method, for building JSON string from a bean

# Questions?