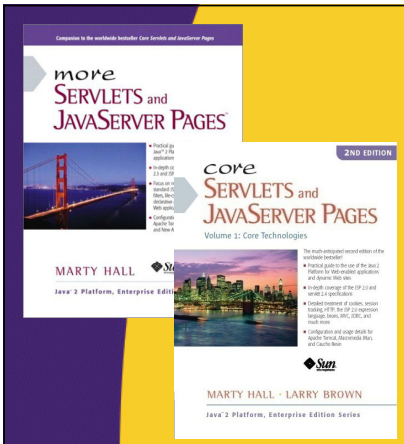
Automatically Generating JSON from Java Objects

Originals of Slides and Source Code for Examples:
http://courses.coreservlets.com/Course-Materials/ajax.html



Topics in This Section

- **Using org.json Java utilities**
  - Building JSON object from bean
  - Building JSON array from Java array or List
  - Building JSON object from Map
  - Other JSON-generation utilities
- **Using json2.js JavaScript utilities**
  - Sending JSON objects *to* server

Intro and Setup

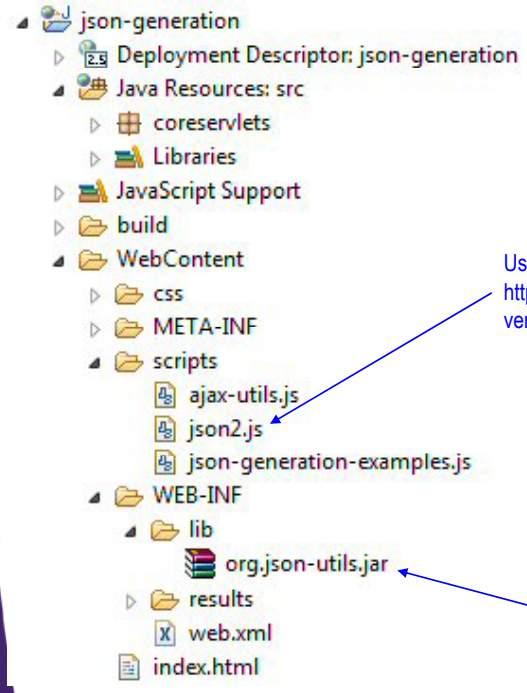
Using MVC to Build JSON

- **Last section: used MVC to build JSON**
  - Advantages
    - Requires no special server software
    - You have full control over result
  - Disadvantages
    - Tedious for complex data structures
    - Often requires knowledge of how server will use data
- **This section: turning Java into JSON**
  - Advantages
    - Can generate complex data easily
    - Builds real objects so server can decide what to do
  - Disadvantages
    - Requires JSON-specific server software
    - Sometimes builds objects with unneeded data in them

Installing the org.json.* Utilities

- **Download**
  - http://www.json.org/java/json.zip
    - Or start at http://www.json.org/java/ and follow link that says “Free source code is available”.
- **Install**
  - Unzip to get org/json/*.java
  - Put into src folder of Eclipse
    - Create new package org.json, then copy files there
  - They do not supply a JAR file, but you could easily build one yourself, then put JAR file in WEB-INF/lib
    - Built org.json-utils.jar and put online at coreservlets.com
- **Documentation**
  - http://www.json.org/java/

Configuring Eclipse Project



Used only in last example. Available for download at http://www.json.org/json2.js. You can also get compressed version; see http://www.json.org/js.html.

Used in all examples. Available for download at http://courses.coreservlets.com/Course-Materials/ajax.html Downloaded latest .java files from http://www.json.org/java/, compiled, and put resultant .class files in JAR file. Or, you can put .java files from http://www.json.org/java/ directly in src/org.json.

Other JSON-Generation Software

- **org.json utilities (used in this tutorial)**
  - Widely used
    - Used within other utilities (e.g., JSON-RPC)
  - Limited power
- **Alternatives**
  - Google Gson
    - Better support for generics
    - http://code.google.com/p/google-gson/
  - JSON Taglib
    - More usable directly from JSP
    - http://json-taglib.sourceforge.net/
  - VRaptor
    - Uses annotations for much of the work
    - http://vraptor.org/ajax.html
  - Many more
    - See “Java” entry at http://json.org/

Supporting Java Code
(Used in All Examples)

Main Bean: City.java

- **Constructor**

```
public City(String name, int timeZone, int pop) {
  setName(name);
  setTimeZone(timeZone);
  setPop(pop);
}
```

- **Getter methods**
  - getName
  - getTime, getTimeZone
    - Assumes server is in US east coasts, subtracts 0-3 hours based on time zone
  - getPop
    - Raw population as an int
  - getPopulation
    - Formatted population as a String with commas

Utilities for Finding Beans: CityUtils.java

- **Map that associates city name with City**

```
private static Map<String,City> biggestAmericanCities =
  new HashMap<String,City>();
```

  - Populate it with largest US cities

- **Lookup functions**

```
public static City getCity(String name) {
  name = name.toUpperCase();
  return(biggestAmericanCities.get(name));
}
```


Utilities for Finding Beans: CityUtils.java Continued

- **Map that associates category of cities with city names**

```
private static Map<String,String[]> cityTypeMap;
```

- **Lookup function**

```
public static List<City> findCities(String cityType) {
  String[] cityNames = cityTypeMap.get(cityType);
  if (cityNames == null) {
    String[] twoCities = { "New York", "Los Angeles" };
    cityNames = twoCities;
  }
  List<City> cities = new ArrayList<City>();
  for(String cityName: cityNames) {
    cities.add(getCity(cityName));
  }
  return(cities);
}
```

Parent Servlet Class: ShowCities.java

```
public abstract class ShowCities extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setHeader("Cache-Control", "no-cache");
    response.setHeader("Pragma", "no-cache");
    response.setContentType("text/javascript");
    List<City> cities = getCities(request);
    outputCities(cities, request, response);
  }

  protected List<City> getCities(HttpServletRequest request) {
    String cityType = request.getParameter("cityType");
    return(CityUtils.findCities(cityType));
  }
```


Parent Servlet Class: ShowCities.java Continued

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

  public abstract void outputCities
                            (List<City> cities,
                             HttpServletRequest request,
                             HttpServletResponse response)
      throws ServletException, IOException;
}
```

General Approach

Steps for Servlet Using JSON Utilities

- **Set normal response headers**
  - response.setHeader for Pragma and Cache-Control
- **Set Content-Type to text/javascript**
  - response.setContentType("text/javascript");
- **Get PrintWriter in normal manner**
  - PrintWriter out = response.getWriter
- **Get result as bean, array, or Map**
  - Call normal business logic code
- **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);

Steps for Using JSON Utilities: Sample Servlet Code

```
public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
    throws ServletException, IOException {
  response.setHeader("Cache-Control", "no-cache");
  response.setHeader("Pragma", "no-cache");
  response.setContentType("text/javascript");
  PrintWriter out = response.getWriter();
  SomeBean javaResult = callSomeBusinessLogic(…);
  JSONObject jsonResult = new JSONObject(javaResult);
  out.print(jsonResult);
}
```

These two lines are the only ones that typically change from application to application. Other lines stay exactly as is.

Turning Java Beans into JSONObject

Turning Beans into JSON

- **org.json defines JSONObject**
  - Its toString method builds JSON format
- **Most important constructor takes a bean**
  - JSONObject json = new JSONObject(myBean);
    - Second arg of "true" means to include superclass info
  - Result
    - Uses reflection on myBean to find all public methods of the form getBlah (any return type) or isBlah (boolean return)
    - Calls each getter method
    - If myBean has getFoo and getBar, it builds object of the form { "foo": "*getFoo() result*", "bar": "*getBar() result*"}
- **Other capabilities**
  - Can turn Map into JSONObject (keys become properties)
  - Can add properties one at a time with "put"

JSONObject from Bean: Example Code

```
package coreservlets;

import org.json.*;

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);
  }
}
```

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)**

Building Arrays of JSON Info

- **org.json defines JSONArray**
  - Its toString method outputs array in JSON format
- **Most important constructors**
  - new JSONArray(javaArrayOrCollection)
    - Assumes javaArrayOrCollection contains primitives, Strings, or JSONObjects
  - new JSONArray(javaArrayOrCollection, false)
    - Assumes javaArrayOrCollection contains beans that should be converted as in previous section, but you don't want to include superclass info
  - new JSONArray(javaArrayOrCollection, true)
    - Assumes javaArrayOrCollection contains beans that should be converted as in previous section, but you do want to include superclass info

JSONArray: Example Code

```
package coreservlets;

import org.json.*;
import java.util.*;

public class CityTest2 {
  public static void main(String[] args) {
    List<City> biggestUSCities =
      CityUtils.findCities("top-5-cities");
    JSONArray citiesJSON =
      new JSONArray(biggestUSCities, false);
    System.out.println("JSON version of biggest " +
                       "US cities is:\n" +
                       citiesJSON);
  }
}
```


JSONArray: Example Result

```
JSON version of biggest US cities is:
[{"time":"09:14:16 AM", "name":"New York",
  "timeZone":0,"pop":8250567,"population":"8,250,567"},
 {"time":"06:14:16 AM", "name":"Los Angeles",
  "timeZone":-3,"pop":3849368,"population":"3,849,368"},
 {"time":"08:14:16 AM", "name":"Chicago",
  "timeZone":-1,"pop":2873326,"population":"2,873,326"},
 {"time":"08:14:16 AM", "name":"Houston",
  "timeZone":-1,"pop":2144491,"population":"2,144,491"},
 {"time":"07:14:16 AM", "name":"Phoenix",
  "timeZone":-2,"pop":1512986,"population":"1,512,986"}]
```

- **(White space added for readability)**

Comparing Manual and Automatic JSON Generation

Manual Generation: Server Code (Servlet)

```
public class ShowCities1 extends ShowCities {
  public void outputCities(List<City> cities,
                           HttpServletRequest request,
                           HttpServletResponse response)
      throws ServletException, IOException {
    request.setAttribute("cities", cities);
    String outputPage =
      "/WEB-INF/results/cities-json.jsp";
    RequestDispatcher dispatcher =
      request.getRequestDispatcher(outputPage);
    dispatcher.include(request, response);
  }
}
```

Manual Generation: Server Code (JSP)

```
{ headings: ["City", "Time", "Population"],
  cities: [["${cities[0].name}", "${cities[0].time}",
            "${cities[0].population}"],
           ["${cities[1].name}", "${cities[1].time}",
            "${cities[1].population}"],
           ["${cities[2].name}", "${cities[2].time}",
            "${cities[2].population}"],
           ["${cities[3].name}", "${cities[3].time}",
            "${cities[3].population}"],
           ["${cities[4].name}", "${cities[4].time}",
            "${cities[4].population}"]]
}
```


Manual Generation: Client Code

```
function cityTable1(address, inputField, resultRegion) {
  var data = "cityType=" + getValue(inputField);
  ajaxPost(address, data,
           function(request) {
             showCityInfo1(request, resultRegion);
           });
}
```

- **Note:**
  - ajaxPost shown in previous tutorial section
    - Sends data via POST and passes result to handler function

Manual Generation: Client Code (Continued)

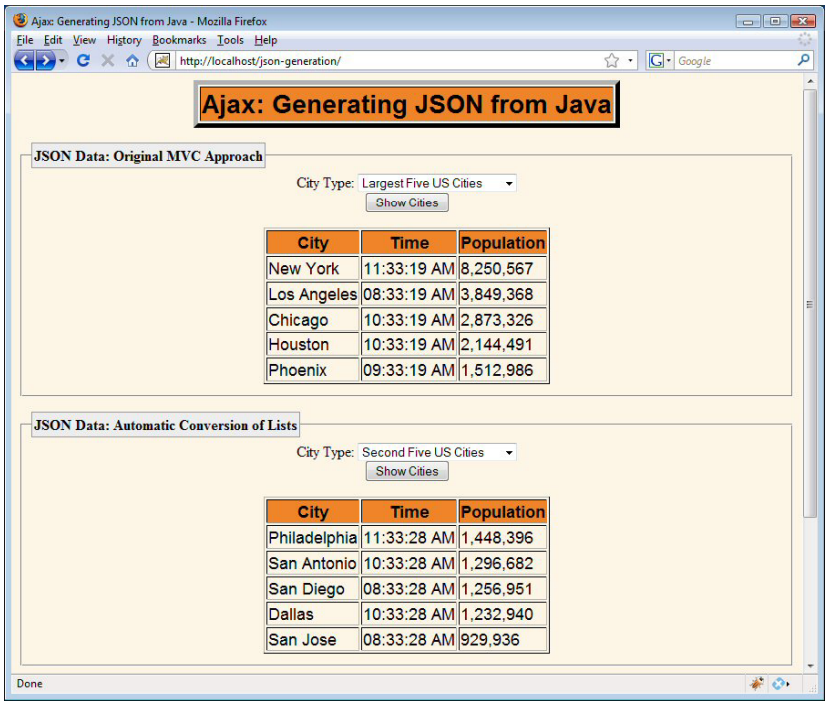
```
// Data that arrives is JSON object with two properties:
//   - headings (an array of strings for the th elements)
//   - cities (an array of array of strings
//              matching the heading names)

function showCityInfo1(request, resultRegion) {
  if ((request.readyState == 4) &&
      (request.status == 200)) {
    var rawData = request.responseText;
    var data = eval("(" + rawData + ")");
    var table = getTable(data.headings, data.cities);
    htmlInsert(resultRegion, table);
  }
}
```


Manual Generation: HTML Code

```
<fieldset>
  <legend>JSON Data: Original MVC Approach</legend>
  <form action="#">
   <label for="city-type-1">City Type:</label>
   <select id="city-type-1">
     <option value="top-5-cities">Largest Five US Cities</option>
     <option value="second-5-cities">Second Five US Cities</option>
     …
    </select>
   <br/>
   <input type="button" value="Show Cities"
          onclick='cityTable1("show-cities-1", "city-type-1",
                              "json-city-table-1")'/>
  </form>
  <p/>
  <div id="json-city-table-1"></div>
</fieldset>
```

Manual Generation: Results



Manual Generation: Pros and Cons

- **Advantages**
  - Requires no JSON-specific software on server
  - Java code is moderately simple
  - Client code is simple
- **Disadvantages**
  - JSP code is complex
  - JSP code cannot adapt to arbitrary number of cities
    - This can be fixed with JSTL – see next tutorial section
  - Server code needs to know a lot about how client code will use results. Server code essentially pre-processed the data and put it in form ready for presentation.
    - If you are going to do that, why bother with data-centric Ajax? Why not just send HTML table from the server?

Automatic Generation: Server Code (Servlet)

```
public class ShowCities2 extends ShowCities {
  public void outputCities(List<City> cities,
                           HttpServletRequest request,
                           HttpServletResponse response)
      throws ServletException, IOException {
    PrintWriter out = response.getWriter();
    out.println(new JSONArray(cities, false));
  }
}
```


Automatic Generation: Server Code (JSP)

- None!

Automatic Generation: Client Code

```
function cityTable2(address, inputField, resultRegion) {
  var data = "cityType=" + getValue(inputField);
  ajaxPost(address, data,
           function(request) {
             showCityInfo2(request, resultRegion);
           });
}
```

- **Note:**
  - Only difference from previous example is that result is passed to showCityInfo2 instead of ShowCityInfo1

Automatic Generation: Client Code (Continued)

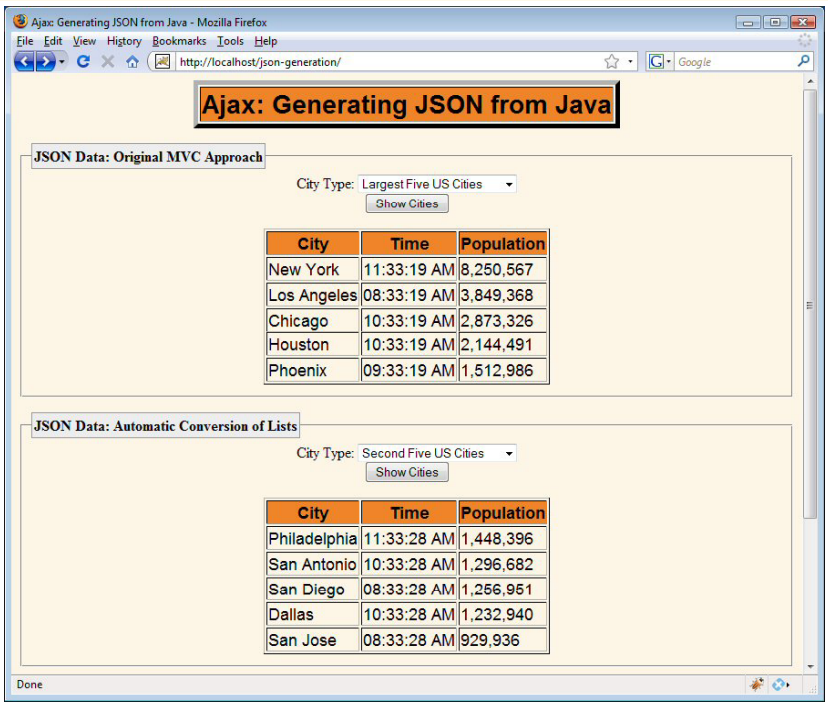
```
// Data that arrives is an array of city objects.
// City objects contain (among other things)
// name, time, and population properties.

function showCityInfo2(request, resultRegion) {
  if ((request.readyState == 4) &&
      (request.status == 200)) {
    var rawData = request.responseText;
    var cities = eval("(" + rawData + ")");
    var headings = ["City", "Time", "Population"];
    var rows = new Array();
    for(var i=0; i<cities.length; i++) {
      var city = cities[i];
      rows[i] = [city.name, city.time, city.population];
    }
    var table = getTable(headings, rows);
    htmlInsert(resultRegion, table);
  }
}
```

Automatic Generation: HTML Code

```
<fieldset>
  <legend>JSON Data: Automatic Conversion of Lists</legend>
  <form action="#">
   <label for="city-type-2">City Type:</label>
   <select id="city-type-2">
     <option value="top-5-cities">Largest Five US Cities</option>
     <option value="second-5-cities">Second Five US Cities</option>
     …
    </select>
   <br/>
   <input type="button" value="Show Cities"
          onclick='cityTable2("show-cities-2", "city-type-2",
                              "json-city-table-2")'/>
  </form>
  <p/>
  <div id="json-city-table-2"></div>
</fieldset>
```


Automatic Generation: Results



Automatic Generation: Pros and Cons

- **Advantages**
  - Java code is very simple
  - No JSP whatsoever
  - Server code can adapt to arbitrary number of cities
  - Server code does not need to know how client code will use the result
  - Client code has "real" data so can do logic based on it
- **Disadvantages**
  - Requires JSON-specific software on server
  - Client code is more complex
    - It needs to extract data from objects before sending it to table-building function
  - Extra fields were sent
    - Client did not use timeZone and pop properties, but they were sent anyway

Turning Java Maps into JSONObject

Building JSONObject from Map

- **Most important JSONObject constructors**
  - new JSONObject(bean)
    - Uses reflection on myBean to find all public methods of the form getBlah (any return type) or isBlah (boolean return)
    - Calls each getter method
    - If myBean has getFoo and getBar, it builds object of the form { "foo": "*getFoo() result*", "bar": "*getBar() result*"}
  - new JSONObject(bean, true)
    - Same as above but includes inherited methods
- **Other constructors**
  - new JSONObject(map)
    - Map keys become JSON property names
  - new JSONObject(string)
    - Useful when passing JSON *to* the server

JSONObject from Map: Example Code

```
package coreservlets;

import org.json.*;
import java.util.*;

public class CityTest3 {
  public static void main(String[] args) {
    Map<String,String[]> cities =
      CityUtils.getCityTypeMap();
    JSONObject citiesJSON =
      new JSONObject(cities);
    System.out.println("JSON version of map of " +
                       "US cities is:\n" +
                       citiesJSON);
  }
}
```

JSONObject from Map: Example Result

```
JSON version of map of US cities is:
{"superbowl-hosts":
  ["Phoenix","Miami",
   "Detroit","Jacksonville","Houston"],
 "top-5-cities":
   ["New York","Los Angeles",
    "Chicago","Houston","Phoenix"],
 "cities-starting-with-s":
   ["San Antonio","San Diego",
    "San Jose","San Francisco","Seattle"],
 "second-5-cities":
   ["Philadelphia","San Antonio",
    "San Diego","Dallas","San Jose"]}
```

- (White space added for readability)

Converting Maps: Server Code

```
public class ShowCityTypes extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setHeader("Cache-Control", "no-cache");
    response.setHeader("Pragma", "no-cache");
    response.setContentType("text/javascript");
    PrintWriter out = response.getWriter();
    JSONObject cityTypes =
      new JSONObject(CityUtils.getCityTypeMap());
    out.println(cityTypes);
  }
```

Converting Maps: Server Code (Continued)

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


Converting Maps: Client Code

```
function cityTypeList(address, resultRegion) {
  ajaxPost(address, null,
           function(request) {
             showCityTypeInfo(request, resultRegion);
           });
}


// Data that arrives is an object where the
// properties are city categories and the
// associated values are arrays of city names.
```

Converting Maps: Client Code (Continued)

```
function showCityTypeInfo(request, resultRegion) {
  if ((request.readyState == 4) &&
     (request.status == 200)) {
    var rawData = request.responseText;
    var cityTypes = eval("(" + rawData + ")");
    var headings = new Array();
    var row1Entries = new Array();
    var i = 0;
    for(var cityType in cityTypes) {
      headings[i] = cityType;
      row1Entries[i] = getBulletedList(cityTypes[cityType]);
      i++;
    }
    var rows = [row1Entries];
    var result = getTable(headings, rows);
    htmlInsert(resultRegion, result);
  }
}
```

Object property names are city categories like "top-5-cities"

Object property values are arrays of city names (cities that match the category)

Converting Maps: Client Code (Continued)

```
function getBulletedList(listItems) {
  var list = "<ul>\n";
  for(var i=0; i<listItems.length; i++) {
    list = list + "  <li>" + listItems[i] + "</li>\n";
  }
  list = list + "</ul>"
  return(list);
}
```

Converting Maps: HTML Code

```
<fieldset>
  <legend>JSON Data: Automatic Conversion of Maps</legend>
  <form action="#">
   <input type="button" value="Show City Types"
          onclick='cityTypeList("show-city-types",
                                "city-types")'/>
  </form>
  <p/>
  <div id="city-types"></div>
</fieldset>
```


Converting Maps: Results

Sending JSON Data from Client to Server

Using JSON.stringify

- **Download json2.js**
  - http://www.json.org/json2.js
  - Or, start at http://www.json.org/js.html and follow links at bottom of page
- **Install in your project**
  - E.g., in Eclipse, drop in WebContent/scripts
  - Load json2.js in your HTML file
- **Call JSON.stringify on JavaScript object**
  - Produces string in JSON format representing object
- **Call escape on result**
  - URL-encode characters for transmission via HTTP
- **Send in POST to client**
  - Data might be large, so POST is better than GET

Utility Function

```
function makeJsonString(object) {
  return(escape(JSON.stringify(object)));
}
```


Receiving JSON Objects on Server

- **Pass string to JSONObject or JSONArray constructor**
  - String jsonString = request.getParameter(…);
  - JSONArray myArray = new JSONArray(jsonString);
- **Access elements with getBlah methods**
  - Primitives
    - getInt, getDouble, getString, getBoolean, isNull
    - double d = myArray.getDouble(0);
      - Server needs to know the types that will be sent from client
  - High-level
    - getJSONObject, getJSONArray

Sending JSON to Server: Client Code

```
function randomCityTable(address, resultRegion) {
  var data = "cityNames=" +
             makeJsonString(getRandomCities());
  ajaxPost(address, data,
           function(request) {
             showCityInfo2(request, resultRegion);
           });
}
```

This is the same showCityInfo2 function used earlier. Takes an array of city objects and makes HTML table from their names, times, and populations.

Sending JSON to Server: Client Code (Continued)

```
var cityNames =
  ["New York", "Los Angeles", "Chicago", "Houston",
   "Phoenix", "Philadelphia", "San Antonio", "San Diego",
   "Dallas", "San Jose", "Detroit", "Jacksonville",
   "Indianapolis", "San Francisco", "Columbus", "Austin",
   "Memphis", "Fort Worth", "Baltimore", "Charlotte",
   "El Paso", "Milwaukeee", "Boston", "Seattle",
   "Washington DC", "Denver", "Louisville", "Las Vegas",
   "Nashville", "Oklahoma City", "Miami"];
```

Sending JSON to Server: Client Code (Continued)

```
function getRandomCities() {
  var randomCities = new Array();
  var j = 0;
  for(var i=0; i<cityNames.length; i++) {
    if(Math.random() < 0.25) {
      randomCities[j++] = cityNames[i];
    }
  }
  return(randomCities);
}
```

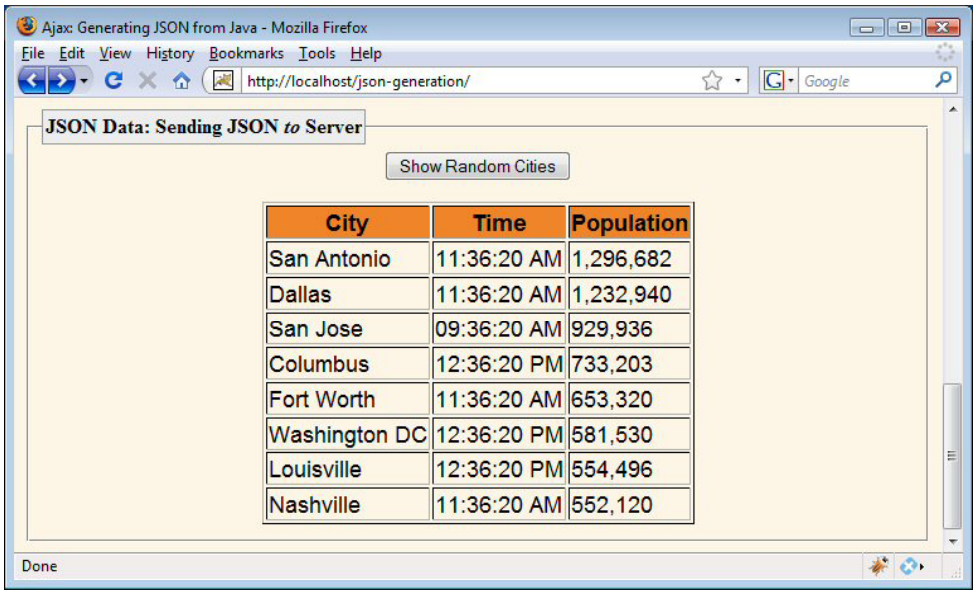

Sending JSON to Server: HTML Code

```
<script src="./scripts/ajax-utils.js"
        type="text/javascript"></script>
<script src="./scripts/json-generation-examples.js"
        type="text/javascript"></script>
<script src="./scripts/json2.js"
        type="text/javascript"></script>
…
<fieldset>
  <legend>JSON Data: Sending JSON <i>to</i>
   Server</legend>
  <form action="#">
   <input type="button" value="Show Random Cities"
          onclick='randomCityTable("show-cities-3",
                                    "json-city-table-3")'/>
  </form>
  <p/>
  <div id="json-city-table-3"></div>
</fieldset>
```

Sending JSON to Server: Server Code

```
public class ShowCities3 extends ShowCities {
  protected List<City> getCities(HttpServletRequest request) {
    String cityNames = request.getParameter("cityNames");
    if ((cityNames == null) || (cityNames.trim().equals("")) {
      cityNames = "['New York', 'Los Angeles]";
    }
    try {
      JSONArray jsonCityNames = new JSONArray(cityNames);
      List<City> cities = new ArrayList<City>();
      for(int i=0; i<jsonCityNames.length(); i++) {
        City city =
          CityUtils.getCityOrDefault(jsonCityNames.getString(i));
        cities.add(city);
      }
      return(cities);
    } catch(JSONException jse) {
      return(CityUtils.findCities("top-5-cities"));
    }
  }
```


Sending JSON to Server: Results



Wrap-up

Preview of Next Section: JSON-RPC

- **Simpler designation of server-side resource**
  - Client-side code acts as if it is calling a server-side function (not a URL)
- **Simpler client-side code**
  - Client-side code passes and receives regular arguments
    - Passing: no need to escape data or build param strings
    - Receiving: no need to use responseText or eval
- **Simpler server-side code**
  - Server-side code receives and returns regular arguments
    - Receiving: no need to call request.getParamter & convert
    - Returning: results automatically converted with JSONObject and JSONArray

Summary

- **Building JSON from Java**
  - new JSONObject(bean)
  - new JSONArray(arrayOrCollectionOfBeans, false)
  - new JSONObject(map)
- **Outputting JSON String**
  - myJSONObject.toString(), myJSONArray.toString()
    - When you do out.print, toString is invoked automatically
- **Sending JSON to server**
  - escape(JSON.stringify(javaScriptObject))
- **Receiving JSON on server**
  - new JSONObject(string) or new JSONArray(string)
  - myArray.getString(i), myArray.getDouble(i), etc.

Questions?