95-733 Internet Technologies August 27, 2013

Homework 1 Three parts

Due: 11:59 PM Tuesday January 28, 2014

This project covers several important topics. The objective of the first part is to introduce the student to AJAX (Asynchronous JavaScript and XML). It also may be used to compare and contrast XML with JSON.

The objective of the second part is to demonstrate that the web may be a source of data for stand alone programs. That is, part 2 does not make use of a browser but still makes good use of the web. It introduces the student to the notion of a grammar for a language and also includes an interesting algorithm that employs backtracking.

The third part makes use of the Google Map API. This part also includes an example of cascading style sheets and makes use of the javascript library provided by Google.

This project is made up of three parts. Each Part is worth 33% of your project 1 score.

Summary of Point Breakdown

Part 1's point breakdown is as follows:

1.1	40%
1.2	40%
1.3	20%

Part 2's point breakdown is as follows:

With a full scheduler using the bear's game or other algorithm, the student may earn up to 100% for part 2. The grammar development piece is also required.

By tackling the simpler problem of simply displaying the remote schedules, the student may earn up to 80% for part 2. The grammar development piece is also required.

Part 3's point breakdown is as follows:

By making an interesting change to the Google Map example, 100% will be awarded for part 3.

By making trivial changes to the Google Map example, 80% will be awarded.

Part 1 Ajax and JSON

0) Read the article by Philip McCarthy on AJAX at:
http://www.andrew.cmu.edu/user/mm6/95-733/Ajax/McCarthyAjaxforJavaDevelopers.html

1) Build a working AJAX shopping cart using the code from the article. The code is found here:

http://www.andrew.cmu.edu/user/mm6/95-733/Ajax/j-ajax1.zip

So that the TA's have an easy time grading the assignments, Netbeans and Glassfish will be used as demonstrated in class.

Name your project ACoolAJAXShoppingCart.

It is a requirement that you employ an external CSS style sheet to add a bit of value to McCarthy's HTML. You may add any style features that you feel would make the presentation more interesting and attractive.

2) Add a button to each item displayed so that the user may delete items from the shopping cart. Your delete button will behave like the add button, but will count down rather than up. If an item is not currently in the cart then a delete request will be ignored. If a delete request causes a quantity to go to zero then the deleted item will be removed from the cart and no longer displayed.

   Name your project ACoolAJAXShoppingCartImproved.

3) Modify your solution to part 2 so that it uses JSON rather than XML. You are required to use the Google JSON libray found here: http://code.google.com/p/json-sans-eval/. Note that your servlet will generate a content type of application/json and your XMLHttpRequest object will return JSON via the responseText property rather than the responseXML property.

   This part requires that you design a JSON string that corresponds to the XML shopping cart. Be sure to see Google's example at the link given above.

   Name your project ACoolAJAXShoppingCartImprovedJSON.

   For a careful description of JSON see http://www.json.org/

Part 1 Submission
=================

Place your work into a single directory named YourNameHomework1Part1. Zip this directory into a file named YourNameHomework1Part1.zip.

Submit your zip file to the Assignment section of Blackboard.

For full credit, you are required to document parts 2 and 3 only.

Part 2 StAX processing and XSD
==============================

There are three standard approaches to processing XML formatted data. These are DOM, SAX and StAX. In this part, we will be using the StAX approach.

JSR 173 defines the Streaming API for XML. This part of homework 1 will use JSR 173 to process schedules and compute a common meeting time. Your solution will include the import of the javax.xml.stream.* API.

There are four schedules found under the directory www.andrew.cmu.edu/~mm6/95-702/McCarthysSchedule. These schedules are named schedule1.xml, schedule2.xml and so on. There is also an XSDL document called schedule.xsd that contains the grammar for the schedule language. Use a browser to examine one of the four schedules and study the schedule grammar carefully.

Write a program in Java called Scheduler that attempts to find a meeting time when n > 1 people are free to meet. Scheduler examines a set of schedules and tries to find a meeting time for each day of the week. If it is able to find a common meeting time then it displays the day and time of the meeting. If it is unable to find such a time it announces that fact for that particular day. It does this for each of the seven days. That is, for each day, a common meeting time is either announced or declared as not possible. The input to the scheduling process will include a minimum meeting time in seconds. If a meeting time of 60 * 60 = 3600 seconds is required then the scheduler will not generate meeting times for anything less than one hour.

Scheduler will read a list of URL's from a local urlList.xml file. It will then fetch an XML document from each of these URL's. It will compute any common meeting times and display a report to the user.

Looking for 2 hour meeting times, when applied to all four schedules, my scheduler produces the following output:

```
java Scheduler 7200
Loading 4 schedules.
**This group can't meet on Monday**
**This group can't meet on Tuesday**
**This group can't meet on Wednesday**
**This group can't meet on Thursday**
**This group can't meet on Friday**
**This group can't meet on Saturday**
**Meeting scheduled for a minimum of 7200 seconds at 13:0:0:16:0:0
on Sunday.
```

The minimum meeting time will be passed to your program via a command line argument.

The above screen dump may not currently be accurate because the XML files have changed.

Suggested Approach
==================

It is suggested that you write an object oriented solution to this problem. Clearly, one type of object that we want to represent is a schedule object. Define a Schedule class with a single constructor and a single accessor method. The constructor for a Schedule object will take a URL object as an input parameter. It will then read the entire document using StAX and extract and retain data. These data will be made available to the user via the second method of this class. This second method will be called getAvailable and will have a signature similar to the following:

```
public LinkedList getAvailable(String day) throws Exception
```

There will be a second class called URLList that is used to represent objects holding URLList documents. Its constructor takes a single URL as an input parameter and uses StAX to read the contents of a URLList document. It retains data and makes these data available via two accessor methods. These methods have the following signatures:

```
    public int getNumURLs() throws Exception

    public String getURL(int i) throws Exception
```

Another class that maintains a start and stop time and provides utility methods for time calculations is also called for. Think about what utility methods you will need.

The scheduling activity might best be written as a static method of the Scheduler class.

As a general rule, don't try to solve the entire problem. Solve smaller problems by hand and focus on building classes and objects that will act as useful tools for the larger problem. Test the classes and objects on smaller instances of the problem. Solve smaller problems first.

Algorithm

In my solution, I used a simple backtracking search as exemplified in Michael Main's "Bear Game". This is the approach that I suggest that you take. In the "bear game", we start with an initial number of bears and wish to reach a goal number of bears within a certain number of steps.

We are only allowed to perform one of two operations to reach our goal. We may increment the number of bears by a fixed constant (provided at run time) or we may divide the current number of bears in half (if the current number of bears is even.)

For example, suppose we start with an initial value of 10 bears and want to reach 5 bears. Suppose too that our increment is 10 and we are allowed to execute 2 steps. Main's algorithm would proceed as follows:

```
    10 --> 20 --> 30 fails need to backtrack
           20 --> 10 because 20 is even but this fails too
       --> 5 we found a solution in one step
```

As an exercise, suppose the goal is 15 bears and we begin with 10 and are allowed 4 steps with an increment of 4.

Here is the code that you might wish to modify to solve the scheduling problem.

```
public static boolean bears (int initial, int goal,
                             int incr, int n)
{
    if (initial == goal)  return true;
    else if (n==0) return false;
    else if (bears(initial+incr, goal, incr, n-1))
           return true;
    else if (initial % 2 == 0)
           return bears(initial/2, goal, incr, n-1);
    else return false;
}
```

Simpler Project (Partial credit)

The most difficult part of the assignment above is the part that requires you to search for a common meeting time using an algorithm like that shown in the "Bears Game". This alternative assignment allows you to skip that part of the problem. Everything else remains the same.

For less credit (maximum of 80%) you may instead write a solution that simply displays all four schedules. The program would still read the URLList.xml file and the schedule files and would still need to perform StAX parsing. The output would look like the following:

```
      java Scheduler
      Loading 4 schedules.
      Schedule1.xml
          Monday
            9:00 - 10:00
           11:00 - 12:00
          Tuesday
            :
      Schedule2.xml
          Monday
            9:00 - 10:00
           11:00 - 12:00
          Tuesday
          :
      :
      Schedule4.xml
      :
```

Developing a simple grammar

It is required that you design an XSDL grammar for the urlList.xml file. See the grammar associated with schedule documents for a guide. Also, you are required to complete the tutorial located at W3C Schools on XML Schema: (see http://www.w3schools.com/schema/default.asp).

Here is a copy of my urlList.xml file. It is currently configured to provide four schedules to the scheduler. Note the reference to the urlList.xsd document.

```
<?xml version="1.0" encoding = "utf-8"?>
<URLList
    xmlns="http://www.andrew.cmu.edu/mm6"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.andrew.cmu.edu/mm6  urlList.xsd"
>
     <URL>http://www.andrew.cmu.edu/~mm6/95-
702/McCarthysSchedule/schedule1.xml</URL>
     <URL>http://www.andrew.cmu.edu/~mm6/95-
702/McCarthysSchedule/schedule2.xml</URL>
     <URL>http://www.andrew.cmu.edu/~mm6/95-
702/McCarthysSchedule/schedule3.xml</URL>
     <URL>http://www.andrew.cmu.edu/~mm6/95-
702/McCarthysSchedule/schedule4.xml</URL>
```

```
</URLList>
```

Use the following program to validate urlList.xml files against the grammar that you write. The grammar must require n > 1 schedule URL's. The program below makes use of Xerces. Information on Xerces and the required jars can be found at the following URL: http://xerces.apache.org/xerces-j/. Currently, as far as I am aware, there is no XSDL validation available with StAX parsers.

Validate.java is a Java program that validates an XML instance against its schema. The schema document (.xsd) must be in the same directory as the document being validated. The document, however, may be pointed to by a URL.

```
// Validate.java using Xerces

import org.xml.sax.InputSource;
import org.xml.sax.SAXParseException;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.helpers.XMLReaderFactory;

public class Validate extends DefaultHandler
{
    public static boolean valid = true;


    public void error(SAXParseException exception) {
         System.out.println("Received notification of a recoverable error." +
exception);
         valid = false;
    }

    public void fatalError(SAXParseException exception) {
         System.out.println("Received notification of a non-recoverable error."+
exception);
         valid = false;
    }

    public void warning(SAXParseException exception) {
          System.out.println("Received notification of a warning."+ exception);
    }


    public static void main (String argv [])
    {
        if (argv.length != 1) {
            System.err.println ("Usage: java Validate [filename.xml | URLToFile]");
            System.exit (1);
        }

        try {
               // get a parser
               XMLReader reader =
XMLReaderFactory.createXMLReader("org.apache.xerces.parsers.SAXParser");

               // request validation
               reader.setFeature("http://xml.org/sax/features/validation",true);

reader.setFeature("http://apache.org/xml/features/validation/schema",true);
               reader.setErrorHandler(new Validate());
               // associate an InputSource object with the file name or URL
               InputSource inputSource = new InputSource(argv[0]);

               // go ahead and parse
               reader.parse(inputSource);
        }
        catch(org.xml.sax.SAXException e) {
              System.out.println("Error in parsing " + e);
              valid = false;

        }
        catch(java.io.IOException e) {
              System.out.println("Error in I/O " + e);
              System.exit(0);
        }
        System.out.println("Valid Document is " + valid);
    }
}
```

Part 2 Submission
=================

Place your work into a single directory named YourNameHomework1Part2. Zip this directory into a file named YourNameHomework1Part2.zip.

Submit your zip file to the Assignment section of Blackboard.

Name your Netbeans project SchedulingWithStax.

Include an XSD file showing the grammar developed for urlList.xml documents.

Submit screenshots showing:

A search for one hour meeting times on the first two schedules (schedule1.xml and schedule2.xml).
A search for one hour meeting time on the third and fourth schedules (schedule3.xml and schedule4.xml).
A search for 30 minute meeting times on all four schedules.

Be able to demonstrate and defend your solution if required.

Part 3 Google Map Mashup
========================

The following code demonstrates the Google Map API and Javascript.

This is a simple demonstration of Google Maps.

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html; charset=UTF-8"/>
      <title>Google Map Example</title>
      <style type="text/css">
          #map1 {
             height: 600px;
             width: 800px;
             float: left;
          }
          #map2 {
             height: 600px;
             width: 800px;
             float: left;
          }
      </style>

      <!-- Get Google's Javascript -->
      <script type="text/javascript" src="http://maps.google.com/maps/api/js?
sensor=false">
      </script>

      <!-- getMap creates a call back handler that executes when the geo coding is
complete.
      -->

      <script type="text/javascript">

      function getMap(s) {

         var userLocation = s.inputbox.value; // location name to geocode

         var pointOfView = { heading:120, pitch:0, zoom:1};

         var geocoder = new google.maps.Geocoder();

         geocoder.geocode(

            {'address': userLocation},

            function(results, status) {

                if (status == google.maps.GeocoderStatus.OK) {

                    //place a Panarama at location map1
                    new google.maps.StreetViewPanorama(
                             document.getElementById("map1"),
                             { position: results[0].geometry.location, pov:
pointOfView, visible:true }
                         );

                    // Prepare to draw street map
                    var mapOptions = {
                                      zoom: 8,
                                      center: results[0].geometry.location,
                                      mapTypeId: google.maps.MapTypeId.ROADMAP
                    };

                    // place a strret map at map2
                    new google.maps.Map(document.getElementById("map2"),mapOptions);

                }
                else {
                    alert("Geocode failed. Reason: " + status);
                }
            }
         );
     }
```

```
    </script>
    </head>
    <body>

        <!-- Ask user for a location - city,state -->
        <!-- Call getMap passing the location to be geocoded and rendered as two 
maps. -->
        <FORM>Enter a location in the box: <BR>
           <INPUT TYPE="text" NAME="inputbox" VALUE=""><P>
           <INPUT TYPE="button" NAME="button" Value="Get Street View" 
onClick="getMap(this.form)">
        </FORM>
       <div id="map1"></div>

       <div id="map2"></div>
    </body>
</html>
```

Part 3 Submission

(1) Make some modifications to the HTML or Javascript shown above so that some interesting additional feature(s) of Google Maps is demonstrated. Submit an HTML file (modified from the one shown above) and a screen scrape of the output. In your submission, within the comment section of the HTML, explain what modifications you made and what you are trying to demonstrate. This modified code should be of real interest to someone trying to learn Google Maps and Javascript. This type of modification is worth 100% for Part 3.

(2) You may instead submit a less elaborate modification. For example, you might add an event handler to the text box so that, when the return key is hit, the map is fetched. In other words, drop the use of the button. In addition, you might use Javascript to detect an empty text book or some other error checking on the text box before calling the geocoding function at Google. Currently, an empty text box may be entered. These types of improvements are worth 80%.