3. Unit: XML Processing with Java (I)

The first four exercises are "typical" exercises to get into the new technologies:

Exercise 3.1 (DOM Basics)

Parse mondial.xml into a DOM instance and implement the following query based on the DOM operations (do not apply XPath in DOM):

For all organisations that have their headquarter in the capital of a member country, output the name of the organisation and the name of the headquarter (to System.out).

```
import java.io.File;
import java.util.List;
import java.util.ArrayList;
import java.util.Map;
import java.util.HashMap;

import org.jdom2.Document;
import org.jdom2.Content;
import org.jdom2.Element;
import org.jdom2.Attribute;
import org.jdom2.Text;
// apt-get install libjdom2-java; add jdom2 to the classpath
import org.jdom2.xpath.XPathFactory;
import org.jdom2.xpath.XPathExpression;
import org.jdom2.input.SAXBuilder;

public class MondialDOM {
  static XPathFactory xpf = XPathFactory.instance();
  static XPathExpression xpath = null;

  public static void xpathtests(Document doc) {
    try {
      // Testbed for XPath expressions:
      // xpath = xpf.compile("//country[@car_code='D']/name");
      // xpath = xpf.compile("//country[@car_code='D']/@capital");      // yes
      // xpath = xpf.compile("id('D')");  // no, nullptr
      // xpath = xpf.compile("//country[@car_code='D']/id(@capital)");  // no, parse syntax
      // xpath = xpf.compile("id(//country[@car_code='D']/@capital)");  // no, nullptr
      xpath = xpf.compile("//country[@car_code='D']//city[name='Berlin']/population[last()]");
      // xpath = xpf.compile("//country[@car_code='D']//city/population[last()]/text()");
      // xpath = xpf.compile("//country[@car_code='D']//city/population[last()]/number()"); // no
      // xpath = xpf.compile("max(//country[@car_code='D']//city/population[last()])"); // no such f
      // xpath = xpf.compile("//country[@car_code='D']//city/population[last()]/string()"); // no
      // Object res = xpath.evaluateFirst(doc);
      // System.out.println("Ergebnis: " + res.toString());
      List<Object> res = xpath.evaluate(doc);
      for (Object r:res) { System.out.println(r.toString()); }
    } catch (Exception e) { e.printStackTrace(); }
  }
  public static void headqquery(Document doc) {
    // for this, it makes sense to think before about the most efficient strategy
    // map organization id->name (for output)
    // map hq-city-id -> list(org-ids)
    // iterate country->capital -> orgs, check for ismember (country@memberships)
    Map<String,List<String>> hqorgs = new HashMap<String,List<String>>();
    Map<String,String> orgnames = new HashMap<String,String>();
    Element mondial = doc.getRootElement();
    for (Element org : mondial.getChildren("organization")) {
      if (org.getAttributeValue("headq") != null) {
         String hq = org.getAttributeValue("headq");
         List<String> orgids;
         orgids = hqorgs.get(hq);
         if (orgids == null) {
```

```
            orgids = new ArrayList<String>();
            hqorgs.put(hq, orgids);
          }
          orgids.add(org.getAttributeValue("id"));
          orgnames.put(org.getAttributeValue("id"),org.getChild("name").getText());
        }
      }
      for (Element country: mondial.getChildren("country")) {
        String membershipsS = country.getAttributeValue("memberships");
        String[] memberships;
        if (membershipsS != null) {
          memberships = membershipsS.split("␣");
          String capitalid =  country.getAttributeValue("capital");
          if (capitalid != null && hqorgs.get(capitalid) != null &&
              memberships.length > 0) {
            for (String orgid : (List<String>) (hqorgs.get(capitalid)))
              for (String ms : memberships)
                if (orgid.equals(ms))
                  System.out.println(orgnames.get(orgid));
          }
        }
      }
    }
    public static void poplisttable(Document doc) {
      System.out.println("<html><table>");
      XPathExpression xpath = xpf.compile("//country//city/population[last()]/text()");
      List<Object> res = xpath.evaluate(doc);
      int[] popnums = new int[10000];  // little bit dirty - better use another loop
      int[] popsums = new int[10000];
      int max = 0;
      for (Object popT: res) {
        int pop = Integer.valueOf(((Text)popT).getTextTrim());
        popnums[pop/10000]++;
        popsums[pop/10000] = popsums[pop/10000] + pop;
        if (pop > max) max = pop;
        }
      for (int i=0; i<=max/10000; i++)
        System.out.println("<tr><td>"+i+"</td><td>" +popnums[i] +"</td>"
                           +   "<td>"+popsums[i]+"</td><tr>");
      System.out.println("</table></html>");
    }
    public static void main(String[] args) {
      try {
        SAXBuilder builder = new SAXBuilder();
        Document mondial = (Document) builder.build(new File("../mondial.xml"));
        //xpathtests(mondial);
        //headqquery(mondial);
        poplisttable(mondial);
        } catch (Exception e) { e.printStackTrace(); }
    }
}
```

Exercise 3.2 (DOM: Creation of a Statistics Table)

Create an HTML table as a JDOM object (and write it finally into a file) that gives, in steps of 100,000, how many cities exist that have between n00.000 and (n+1)00.000 inhabitants, and how many inhabitants each of these groups has. Note: also if there is no city in such a step, return a line with a "0".

0 - 100000	275	16120153
100000 - 200000	1074	154777116
200000 - 300000	556	134546479
:	:	:
22200000-22300000	0	0
22300000-22400000	1	22315474

For computing the counts and numbers from Mondial, XPath can be used (e.g., read Mondial into a JDOM object and apply XPath to it; you can also use any other package for this if you want).

Where would be a problem to create the same table in XQuery or XSLT?

- Code: see above exercise.
- Same in XQuery/XSLT: how to generate the enumeration 0...22400000 including the steps where no population exists?
  In XQuery, this is possible with
  ```
  for $i in (0 to (round(max(//city/population) div 10000) cast as xs:integer))
  return $i
  ```
  Note: without the cast, it would be xs:double, and the "to" won't work.
  In XSLT, the same can be used:
  ```
  <xsl:for-each select="0 to 5"> x </xsl:for-each>
  ```

Exercise 3.3 (SAX: Queries against Mondial)

Write SAX Event Handlers in Java for the following tasks:

a) Output an HTML file that lists the names of all countries in mondial.xml.

b) Output the name and the population of the capital of Germany via System.out.

c) use the previous part of the exercise to output all country names, the country's capital and the country capital's population – if available – into an HTML table.
Example:

country	**capital**	**capital population**
Albania	Tirane	192000
Greece	Athens	885737
:	:	:

- First, simply output the table values as text to System.out.
- Create an HTML table as a JDOM object with the result during the SAX run and output it into a file.

d) For each country in mondial.xml, output an HTML table containing the names and – if present – the most recent population count for each city in the country. Use a <ul> (unordered list) environment with one list item per country.
Example:

- ...
- **Germany**

Stuttgart	588482
Mannheim	316223
Karlsruhe	277011
...	...

- ...

e) Modify the event handler of the previous part of the exercise to output the following for each country with at least than 10 valid city population entries:

- the country's name

- the overall number of cities
- each city with name and most recent population count
- the average city population
- inside the city table, mark (either by color, font etc) (1) the capital city, and (2) the city whose population is closest to the average city population of the country.

f) Modify the event handler from 2 as follows:

- stop after the population number of Germany's capital has been printed;
- add appropriate additional output to show that the process stopped there;
- instead of printing Germany's population number, print the whole contents of the first argument of the respective `characters(char[], int, int)` call. Explain the result.

```
/* first, provide the main method invoking the sax parser */

import java.io.IOException;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class CallParser {

    public static void main(String[] args) {
        if (args.length != 2) {
            System.err.println("usage: CallParser <uri> <outputfilename>");
            System.exit(1);
        }
        String uri = args[0];
        String outputfilename = args[1];
        DefaultHandler handler = null;
        // handler = new ContentHandlerCountryNames(outputfilename);
        // handler = new ContentHandlerPopBerlin();
        // handler = new ContentHandlerCitySizes(outputfilename);
        // handler = new ContentHandlerCapitalSizes(outputfilename);
        handler = new ContentHandlerCitiesStatistics(outputfilename);
        SAXParserFactory factory = SAXParserFactory.newInstance();
        try {
            SAXParser parser = factory.newSAXParser();
            parser.parse(uri, handler);
        } catch (Exception e1) {
            // my own exception, thrown by the ContentHandlerPopBerlin Handler:
            if (e1 instanceof ContentHandlerPopBerlin.MySAXTerminatorException)
                System.out.println("ready");
            else e1.printStackTrace();
        }
    }
}
```

```
/* content handler for (a) */
import java.io.*;
import java.util.Stack;

import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class ContentHandlerCountryNames extends DefaultHandler {

    Writer out = null;
```

```
    private void put(String s) {
        try { out.write(s); } catch (IOException e) { e.printStackTrace();  }
    }

    public ContentHandlerCountryNames(String outputfilename) {
        try {
            out = new OutputStreamWriter(new FileOutputStream(outputfilename));
        } catch (FileNotFoundException e) { e.printStackTrace(); }
    }

    public void startDocument() throws SAXException {
        String header = "<HTML><HEAD><TITLE>Countries</TITLE></HEAD><BODY><UL>";
        put(header);
    }

    public void endDocument() throws SAXException {
        String footer = "</UL></BODY></HTML>";
        put(footer);
        try { out.close(); } catch (IOException e) { e.printStackTrace(); }
    }

    public void startElement(String uri, String localName, String qName,
            Attributes attrs) throws SAXException {
        stack.push(qName.toLowerCase());
    }

    public void endElement(String url, String localName, String qName)
            throws SAXException {
        stack.pop();
    }

    public void characters(char[] text, int from, int length)
            throws SAXException {
        String textString = (new String(text)).substring(from, from + length);
        if (((String) stack.peek()).equals("name")) {
            stack.pop();
            if (((String) stack.peek()).equals("country")) {
                // found country name
                put("<b>" + textString + "</b><br/>");
            }
            stack.push("name");
        }
    }
}
```

```
/* content handler for (b) */
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class ContentHandlerPopBerlin extends DefaultHandler {
    private String lastElement = "";
    private String capitalID = "";
    private boolean isGermany = false;
    private boolean isdecapital = false;
    private boolean incappop = false;
    private boolean dostop = false;
    private int screenpos = 0;
    private int population = 0;

    public class MySAXTerminatorException extends SAXException {
        ;
    }
```

```
    public void startDocument() throws SAXException {}
    public void endDocument() throws SAXException {}

    public void startElement(String uri, String localName, String qName,
            Attributes attrs) throws SAXException {
        lastElement = qName;
        if (qName.equals("country")) {
            capitalID = attrs.getValue("capital");
            String car_code = attrs.getValue("car_code");
            if (car_code != null && car_code.equals("D")) {
                isGermany = true;
            }
        }
        String id = attrs.getValue("id");
        if (qName.equals("city") && isGermany && id != null
                && id.equals(capitalID))
            isdecapital = true;
        if (qName.equals("population") && isGermany && isdecapital)
           incappop = true;
    }

    public void endElement(String url, String localName, String qName)
            throws SAXException {
        if (qName.equals("city")) {
             if (isdecapital) {
                System.out.print("␣" + population);
                isdecapital = false;
                dostop = true;
             }
        }
        if (qName.equals("country")) {
          isGermany = false;
          System.out.print(".␣");
          screenpos += 2;
        }
    }

    public void characters(char[] text, int from, int length)
            throws SAXException {
        String textString = (new String(text)).substring(from, from + length);
        if (lastElement.equals("name") && isdecapital) {
            // if we are inside the name element of germany's capital
            System.out.print("␣" + textString + "␣");
            screenpos += textString.length() + 2;
        }
        else if (lastElement.equals("population") && isdecapital) {
            // a population element in Berlin
            population = Integer.parseInt(textString);
            screenpos += textString.length() + 2;
        }
        else if (dostop == true) {
            System.out.print("\n" + new String(text));  // output the most recent char[]
            // shows that this are 2048 chars read from the XML input stream,
            // where the from/length addresses just the current text node
            throw new MySAXTerminatorException();
        }
    }
}
```

```
/* content handler for (c)
   note: multiple population entries  */

import java.io.*;
import java.util.Stack;
```

```
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class ContentHandlerCitySizes extends DefaultHandler {
    Writer out = null;
    private Stack stack = new Stack();
    private String name = null;
    private String population = null;

    private void put(String s) {
        try { out.write(s); } catch (IOException e) { e.printStackTrace(); }
    }

    public ContentHandlerCitySizes(String outputfilename) {
        try {
            out = new OutputStreamWriter(new FileOutputStream(outputfilename));
        } catch (FileNotFoundException e) { e.printStackTrace(); }
    }

    public void startDocument() throws SAXException {
        String header = "<HTML><HEAD><TITLE>City␣Sizes</TITLE></HEAD><BODY><UL>";
        put(header);
    }

    public void endDocument() throws SAXException {
        String footer = "</UL></BODY></HTML>";
        put(footer);
    }

    public void startElement(String uri, String localName, String qName,
            Attributes attrs) throws SAXException {
        // country element found
        if (qName.equals("country")) {
            put("<li><table␣border='1'>\n");
        }
        stack.push(qName.toLowerCase());
    }

    public void endElement(String url, String localName, String qName)
            throws SAXException {
        if (qName.equals("country")) {
            put("</table>\n</li>\n");
          } else if (qName.equals("city")) {
            put("<tr>");
            put("<td>");
            if (name != null) put(name);
            put("</td>");
            put("<td>");
            if (population != null) put(population);
            put("</td>");
            put("</tr>\n");
            name = null;
            population = null;
        } else if (qName.equals("name")) {
            stack.pop();
            if (((String)stack.peek()).equals("city")) put("</td>");
            stack.push("name");
        } else if (qName.equals("population")) {
            stack.pop();
            if (((String)stack.peek()).equals("city")) put("</td>");
            stack.push("population");
        }
        stack.pop();
```

```
    }

    public void characters(char[] text, int from, int length)
            throws SAXException {
        String textString = (new String(text)).substring(from, from + length);
        if (((String) stack.peek()).equals("name")) {
            stack.pop();
            if (((String) stack.peek()).equals("country")) {
                // found country name
                put("<b>" + textString + "</b>\n");
            } else if (((String) stack.peek()).equals("city")) {
                // found city name
                name = textString;
            }
            stack.push("name");
        } else if (((String) stack.peek()).equals("population")) {
            stack.pop();
            if (((String) stack.peek()).equals("city")) {
                // found city pop
                population = textString;
            }
            stack.push("population");
        }
    }
}
```

```
/* content handler for (d) */
import java.io.*;
import java.util.Stack;

import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class ContentHandlerCapitalSizes extends DefaultHandler {
    Writer out = null;
    private Stack stack = new Stack();
    private String capitalRef = null;
    private String countryname = null;
    private String capname = null;
    private String population = null;

    private void put(String s) {
        try { out.write(s); } catch (IOException e) { e.printStackTrace(); }
    }

    public ContentHandlerCapitalSizes(String outputfilename) {
        try {
            out = new OutputStreamWriter(new FileOutputStream(outputfilename));
        } catch (FileNotFoundException e) { e.printStackTrace(); }
    }

    public void startDocument() throws SAXException {
        String header =
          "<HTML><HEAD><TITLE>Countries␣and␣their␣capitals</TITLE></HEAD><BODY><UL>"
        + "<table␣border='1'><th>country</th><th>capital</th><th>capital␣population</th>";
        put(header);
    }

    public void endDocument() throws SAXException {
        put("</table></UL></BODY></HTML>");
        try { out.close(); } catch (IOException e) { e.printStackTrace(); }
    }
```

```
    public void startElement(String uri, String localName, String qName,
            Attributes attrs) throws SAXException {
        if (qName.equals("country")) {
            // store capital-reference value
            capitalRef = attrs.getValue("capital");
            stack.push("country");
        } else if (qName.equals("city")
                && attrs.getValue("id").equals(capitalRef)) {
            stack.push("capital");
        } else
            stack.push(qName.toLowerCase());
    }

    public void endElement(String url, String localName, String qName)
            throws SAXException {
        if (qName.equals("country")) {
            put("<tr>");
            put("<td>" + countryname + "</td>");
            countryname = null;
            put("<td>" + capname + "</td>");
            capname = null;
            if (population != null)
               put("<td>" + population + "</td>");
            population = null;
            put("</tr>");
        }
        stack.pop();
    }

    public void characters(char[] text, int from, int length)
            throws SAXException {
        String textString = (new String(text)).substring(from, from + length);
        if (stack.size() > 1) {
            String parent = (String) stack.pop();
            String grandparent = (String) stack.peek();
            if (grandparent.equals("country") && parent.equals("name"))
                countryname = textString;
            if (grandparent.equals("capital") && parent.equals("name"))
                capname = textString;
            if (grandparent.equals("capital") && parent.equals("population"))
                population = textString;
            stack.push(parent);
        }
    }
}
```

```
/* content handler for (e) */
import java.io.*;
import java.util.Stack;
import java.util.Vector;

import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

import mondial.*;

public class ContentHandlerCitiesStatistics extends DefaultHandler {
    Writer out = null;
    private Stack stack = new Stack();
    private MondialCountry country;
    private String countryName = "";
    private MondialCity city;
    private String cityName = "";
```

```
private int cityPopulation = 0;
private Vector countries = new Vector();
private String capitalRef = "";

private void put(String s) {
    try { out.write(s); } catch (IOException e) { e.printStackTrace(); }
}

public ContentHandlerCitiesStatistics(String url, String outputfilename) {
    try {
        out = new OutputStreamWriter(new FileOutputStream(outputfilename));
    } catch (FileNotFoundException e) { e.printStackTrace(); }
}

public void startDocument() throws SAXException {
    put("<HTML><HEAD><TITLE>Cities␣statistics</TITLE></HEAD><BODY>");
}

public void endDocument() throws SAXException {
    put( processCountries() + "</BODY></HTML>");
    try { out.close(); } catch (IOException e) { e.printStackTrace(); }
}

private String processCountries() {
  StringBuffer ret = new StringBuffer("<ol>");
  for (int i = 0; i < countries.size(); i++) {
    country = (MondialCountry) countries.get(i);
    int popCities = 0;
    int sumCityPop = 0;
    // iterate through all countries
    for (int j = 0; j < country.noOfCities(); j++) {
      // calculate sum city pop
      city = country.getCity(j);
      if (city.getPopulation() > 0) {
        popCities++;
        sumCityPop += city.getPopulation();
      }
    }
    // but select only those with >=10 cities with pop data
    if (popCities >= 10) {
      // calculate average pop and minimum difference
      double averagePop = sumCityPop / (double) popCities;
      int posNearest = -1;
      double minDiff = country.getCity(0).getPopulation() - averagePop;
      if (minDiff < 0) minDiff = - minDiff;
      for (int j = 0; j < country.noOfCities(); j++) {
        city = country.getCity(j);
        if (city.getPopulation() > 0) {
          double diff = (city.getPopulation() - averagePop);
          if (diff < 0) diff = -diff;
          if (diff < minDiff) {
            minDiff = diff;
            posNearest = j;
          }
        }
      }
      ret.append("<li><b>" + country.getName() + "</b>");
      ret.append("<table␣border='1'>\n␣␣<tr><td><i>no␣of␣cities</td></i>"
                 + "<td><i>" + country.noOfCities() + "</td></i></tr>");
      ret.append("␣␣<tr><td><i>average␣pop</td></i><td><i>"
                 + (int) averagePop + "</td></i></tr>\n");
      ret.append("␣␣<th>city␣name</th><th>city␣pop</th>\n");
      for (int j = 0; j < country.noOfCities(); j++) {
        city = country.getCity(j);
        String color = "";
```

```
            String markupLeft = "";
            String markupRight = "";
            String pop = "-";
            if (city.getPopulation() > 0)
               pop = Integer.toString(city.getPopulation());
            if (j == posNearest)
               color = "␣bgcolor='lightgray'";
            if (city.isStateCap()) {
               markupLeft = "*";
               markupRight = "*";
            }
            ret.append("␣␣<tr><td" + color + ">" + markupLeft + city.getName()
                  + markupRight + "</td><td␣" + color + ">" +pop+ "</td></tr>\n");
         }
         ret.append("</table></li>");
      }
   }
   ret.append("</ol>");
   return ret.toString();
}

public void startElement(String uri, String localName, String qName,
        Attributes attrs) throws SAXException {
    if (qName.equals("country")) {
        country = new MondialCountry();
        capitalRef = attrs.getValue("capital");
    } else if (qName.equals("city")) {
        city = new MondialCity();
        if (attrs.getValue("id").equals(capitalRef)) {
            city.setStateCap(true);
        } else {
            city.setStateCap(false);
        }
    }
    stack.push(qName.toLowerCase());
}

public void endElement(String url, String localName, String qName)
        throws SAXException {
    if (stack.size() > 1) {
        String parent = (String) stack.pop();
        String grandparent = (String) stack.peek();
        if ((parent.equals("name")) && grandparent.equals("country")) {
            // collect country name
            country.setName(countryName);
            countryName = "";
        } else if (parent.equals("country")) {
            // collect country
            countries.add(country);
        } else if (parent.equals("name") && grandparent.equals("city")) {
            // collect city name
            city.setName(cityName);
            cityName = "";
        } else if (parent.equals("population")
                && grandparent.equals("city")) {
            // collect city pop
            city.setPopulation(cityPopulation);
        } else if (parent.equals("city")) {
            // collect city
            country.addCity(city);
        }
    } else
        stack.pop();
}
```

```
     public void characters(char[] text, int from, int length)
            throws SAXException {
        String textString = (new String(text)).substring(from, from + length);
        if (stack.size() > 1) {
            String parent = (String) stack.pop();
            String grandparent = (String) stack.peek();
            MondialCountry country;
            if (parent.equals("name") && grandparent.equals("country")) {
                countryName += textString;
            } else if (parent.equals("name") && grandparent.equals("city")) {
                cityName += textString;
            } else if (parent.equals("population")
                    && grandparent.equals("city")) {
                cityPopulation = Integer.parseInt(textString);
            }
            stack.push(parent);
        }
    }
}
```

```
public class MondialCity extends MondialObject {
    private int population;
    private boolean isStateCap = false;
    public MondialCity(String name) { super(name); }
    public MondialCity() { super(); }
    public int getPopulation() { return population; }
    public void setPopulation(int population) { this.population = population; }
    public boolean isStateCap() { return isStateCap; }
    public void setStateCap(boolean isStateCap) { this.isStateCap = isStateCap; }
}
```

```
import java.util.Vector;

public class MondialCountry extends MondialObject {
    private Vector cities = new Vector();
    public MondialCountry(String name) { super(name); }
    public MondialCountry() { super(); }
    public void addCity(MondialCity city) { cities.add(city); }
    public MondialCity getCity(int index) { return (MondialCity) cities.get(index); }
    public int noOfCities() { return cities.size(); }
}
```

```
public abstract class MondialObject {
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public MondialObject(String name) { super(); this.name = name; }
    public MondialObject() { super(); }
    private String name;
}
```

Exercise 3.4 (StAX: Queries against Mondial)

a) Implement parts a) and e) as given in the SAX exercise, now by using StAX.
Reuse the program code as far as possible, and adapt it only to the StAX frame.

b) Implement the following query in StAX: For all organisations that have their headquarter in the capital of a member country, output the name of the organisation and the name of the headquarter city.

c) Playing with streams: implement a pipe such that (b) creates results of the form

```
<result>
 <organization name="European Union"/>
 <city name="Brussels"/>
</result>
```

that are written into another XMLOutputStream. Create a second thread that reads from this stream and filters (via StAX) only the <city> elements. Let both threads log to System.out to show the concurrent processing.

```
import java.io.FileInputStream;
import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLStreamConstants;
import javax.xml.stream.XMLStreamReader;
import java.util.Set;
import java.util.TreeSet;
import java.util.Map;
import java.util.HashMap;

public class MondialStaxHQCaps {

  public static void main(String[] args) {
  try {
    // for this, it makes sense to think before about the most efficient strategy
    // collect capital-id->name, -> country, country->set(memberships-org-ids)
    // iterate organizations, compare cap=hq->country and check membership
    Map<String,String> capnames = new HashMap<String,String>();
    Map<String,String> capcountries = new HashMap<String,String>();
    Map<String,Set<String>> memberships = new HashMap<String,Set<String>>();

    FileInputStream inputStream;
    XMLInputFactory inputFactory = XMLInputFactory.newInstance();
    XMLStreamReader parser =
       inputFactory.createXMLStreamReader(new FileInputStream("mondial.xml"));

    Boolean incountry = false;
    Boolean incity = false;
    Boolean incapital = false;
    String capitalid = null;
    Boolean toOutput = false;
    String hq = null;
    Boolean goOn = true;
    int count = 0;
    while (goOn) {
      int event = parser.next();
      switch(event) {
      case XMLStreamConstants.END_DOCUMENT: // ignore. Will stop with first sea.
            break;
      case XMLStreamConstants.START_ELEMENT:
        if ("country".equals(parser.getLocalName())) {
           incountry = true;
           String code = parser.getAttributeValue(null, "car_code");
           capitalid = parser.getAttributeValue(null, "capital");
           capcountries.put(capitalid, code);
           TreeSet<String> mymembers = new TreeSet<String>();
           String orgs = parser.getAttributeValue(null, "memberships");
           if (orgs != null)
             for (String s : orgs.split("␣"))
               mymembers.add(s);
           memberships.put(code, mymembers);
        }
        else if ("city".equals(parser.getLocalName())) {
          incity = true;
```

```
          if (capitalid != null
              && capitalid.equals(parser.getAttributeValue(null, "id")))
            incapital = true;
        }
        else if ("name".equals(parser.getLocalName())) {
          if (incapital) {
            parser.next();
            capnames.put(capitalid, parser.getText());
            incapital = false; // to skip further names of it
          }
          if (toOutput) { // org must be output.
            count++;
            parser.next();
            System.out.println(parser.getText() + ": " + capnames.get(hq));
          }
        }
        else if ("organization".equals(parser.getLocalName())) {
          String id = parser.getAttributeValue(null, "id");
          hq = parser.getAttributeValue(null, "headq");
          String country = capcountries.get(hq);
          if (country != null) { // hq is a capital of a country
            if (memberships.get(country).contains(id))
              toOutput = true;
          }
        }
        else if ("sea".equals(parser.getLocalName()))
            goOn = false;
          break;
      case XMLStreamConstants.CHARACTERS: // ignore. always actively pulled.
          break;
      case XMLStreamConstants.END_ELEMENT:
          if ("country".equals(parser.getLocalName()))
            capitalid = null;
          else if ("city".equals(parser.getLocalName())) {
            incity = false;
            incapital = false;
          }
          else if ("organization".equals(parser.getLocalName())) {
            toOutput = false;
          }
        }
      }
      parser.close();
      System.out.println(count + " Results");
    } catch (Exception e) { e.printStackTrace(); }
    }
}
```

Usecase Scenario: Calexit.

The second group of exercises implement a "realistic" use case for an update to Mondial (recall that XQuery and XSLT cannot be used to *change* a document directly).

Consider the case that California leaves the U.S.A. Then, some updates have to be executed on Mondial:

- plan first your strategy, before starting to program,
- consider especially, what kinds of data items must be changed, and how,
- write the programs as generic as possible (such that they can be applied also whenever some other province turns into a new, independent country).
- Update mondial.xml appropriately, especially:

- take as much data as possible from Mondial,
- California becomes a member of all organizations where the USA are a member, except "Group of 5" and "Group of 7".
- Ignore the airport elements at the end of Mondial.
- The below fragment contains all necessary new facts about California [Filename: caldata.xml]:

```
<country>
  <population_growth>1.0</population_growth>
  <infant_mortality>4.5</infant_mortality>
  <gdp_total>1810000</gdp_total>
  <gdp_agri>2</gdp_agri>
  <gdp_ind>23</gdp_ind>
  <gdp_serv>75</gdp_serv>
  <inflation>1.8</inflation>
  <unemployment>5.5</unemployment>
  <ethnicgroup percentage="72.9">European</ethnicgroup>
  <ethnicgroup percentage="6.5">African</ethnicgroup>
  <ethnicgroup percentage="14.7">Asian</ethnicgroup>
  <ethnicgroup percentage="1.7">Amerindian</ethnicgroup>
  <religion percentage="32">Protestant</religion>
  <religion percentage="28">Roman Catholic</religion>
  <religion percentage="2">Jewish</religion>
  <religion percentage="2">Buddhist</religion>
  <religion percentage="1">Mormon</religion>
  <religion percentage="1">Muslim</religion>
  <language percentage="60.5">English</language>
  <language percentage="25.8">Spanish</language>
  <language percentage="2.6">Chinese</language>
  <border country="MEX" length="226"/>
  <border country="USA" length="1681"/>
</country>
```

The task should be solved by different approaches:

- JDOM: just updating the existing Mondial data [lengthy Java programming?],
- XSLT: as a high-level transformation [declarative - maybe the shortest solution?]
- SAX/StAX: Streaming [the only way for really laaaarge XML data - not just hacking, but a strategy is needed]
- JAXB: Mapped to an object-oriented model [and then same as DOM?]

Hints:

- in the JDOM case, you must explicitly modify some things in MONDIAL; all others stay unchanged.
- For XSLT and SAX/StAX, you have also to deal with the unchanged portions. Handle this with as little effort as possible, using general rules (XSLT templates and SAX/StAX conditions). Note that these two strategies are closely related.
- Focus on the general, structural issues; don't spend too much time for the dirty details of the new values for the USA.
- My solution in XSLT just contains 10 templates (85 lines) for the general handling, and another 8 templates (80 lines) for dealing with a reasonable computation of the new USA values.

Solution: Strategy: Proceed as generic as possible.
This can be best explained by the XSLT reference solution.

The XSLT solution consists of a generic match="*" pattern that just recursively copies all attributes, and text or element children. Then, more specific templates are used that override this rule wherever necessary:

- two generic recursive templates for copying unchanged (elements with attrs, text).
- template/@match="country[@car_code='USA']":
  Handle USA (als all its provinces except CA), then just also process CA.
- template/@match="province[@country='USA' and name='California']": handle CA.
- template/@match="city[@country='USA' and id(@province)/name='California']":
  handle CA cities.
- template/@match="border[../@car_code=$calData/border/@country and @country='USA']":
  borders between neighbors (MEX) of CA and USA change their length (MEX-USA) and add MEX-CA border.
- template/@match="[located[@country='USA' and id(@province)/name='California']]":
  handle all geo things that are located in CA. Process the <located>recursively.
- template/@match=
  "located[@country='USA' and id(@province)/name='California' and count(id(@province))=1]":
  nothing, since CA does not have provinces.
- template/@match=
  "located[@country='USA' and id(@province)/name='California' and count(id(@province))>1]":
  keep the remaining US-provinces in the list.
- template/@match="members[not (../name=('G-5','G-7')) and id(@country)/@car_code='USA']":
  for these organizations, add CA.
- for details of US attributes: templates for US - CA, US - CA weighted by population, US - CA weighted by GDP, apply appropriately (see XSLT code).

Check Results:

- new country CA: must contain cities directly.
  Note that it is *not* necessary to change their IDs (they can still be called cty-US-*nn*), this is just some (internal) identifier.
- CA/indep_date, CA/government, CA/encompassed, CA/@memberships (no G-5, G-7)
- City of Sacramento: is_state_cap="yes".
- CA Cities: country=CA; no province attribute
- USA:
  - populations minus CA: 1980: 206508459, 2014: 280054556E8
  - popgrowth 0.739, infant mortality 6.40, unemployment 7.55 (adapted weighted by population)
  - GDP total 1.491E7; agri 0.99, ind: 19.075, serv: 79.93, infl 1.46 weighted by total gdp
  - Ethnic groups, religions, languages weighted by population: Europ 80.93, African 13.72, Asian 3.00, ... English 85.09, Spanish 8.61
  - border to MEX 3100km, border to CA 1681km
- MEX (i.e. all USA-borders of countries that border CA): MEX-USA 3100km
- geo things: use generic template dependent on @country and located/@province:
  - Mt. Whitney: now only in CA (no <located>)

  - Boundary Peak, Amargosa River: CA and USA; single <located> for USA/Nevada (id 29)
  - Colorado River: MEX, USA, CA; <located> for 4 USA provs and 2 MEX provs.
  - Pacific Ocean: +CA, <located> for 4 USA provs (OR, WA, AL, HI)
- Organizations: G-5, G-7: USA, but no CA; NATO: USA+CA.

Exercise 3.5 (Calexit in JDOM)

- read mondial.xml into a JDOM object,
- update it using the JDOM operations (you can use XPath in the JDOM for searching for values or nodes),
- write it out into a file,
- validate the result file against mondial.dtd.

```
import java.io.InputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.OutputStream;
import java.io.IOException;
import java.io.FileNotFoundException;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;
import java.util.LinkedList;
import java.util.Iterator;
import java.util.Map;

import org.jdom2.Document;
import org.jdom2.Content;
import org.jdom2.Element;
import org.jdom2.Attribute;
import org.jdom2.Text;
import org.jdom2.DocType;
import org.jdom2.JDOMException;
import org.jdom2.xpath.XPathFactory;
import org.jdom2.xpath.XPathExpression;
import org.jdom2.input.SAXBuilder;
import org.jdom2.output.XMLOutputter;

public class CalexitDOM {

  FileInputStream inputStream;
  OutputStream outputStream;
  Document caldata = null;
  Element california = null;
  Element usa = null;
  int calGDP = 0;
  int usGDP = 0;
  int calpop = 0;
  int uspop = 0;
  XPathFactory xpf = XPathFactory.instance();
  XPathExpression xpath = null;

  public CalexitDOM(FileInputStream in, OutputStream out) {
    this.inputStream = in;
    this.outputStream = out;
  }

  public void doIt() {
    SAXBuilder builder = new SAXBuilder();
    File mondialFile = new File("mondial.xml");
```

```
File caldataFile = new File("caldata.xml");
Document mondial = null;
Document caldata = null;
int index;
try {
  mondial = (Document) builder.build(mondialFile);
  Element mondialEl = mondial.getRootElement();
  caldata = (Document) builder.build(caldataFile);

  // turn CAL into a <country> Element
  xpath = xpf.compile("//country[@car_code='USA']/province[name='California']");
  california = (Element) xpath.evaluateFirst(mondial);
  String calID = california.getAttributeValue("id");
  california.removeAttribute("id");
  california.removeAttribute("country");
  usa = california.getParentElement();
  index = mondialEl.indexOf(usa);
  california.detach();
  california.setName("country");
  california.setAttribute("car_code","CA");
  Element areaEl = california.getChild("area");
  areaEl.detach();
  Float calarea = Float.valueOf(areaEl.getTextTrim());
  california.setAttribute("area", calarea.toString());
  String memberships = usa.getAttributeValue("memberships");
  String s = "";
  for (String orgS : memberships.split("␣")) {
    xpath = xpf.compile("//organization[@id='" + orgS + "']");
    Element org = (Element) xpath.evaluateFirst(mondial);
    if (!"Group␣of␣5".equals(org.getChild("name").getTextTrim())
    && !"Group␣of␣7".equals(org.getChild("name").getTextTrim()))
    s += org.getAttributeValue("id") + "␣";
    xpath = xpf.compile("members[contains(@country,␣'USA')]");
    Element members = (Element) xpath.evaluateFirst(org);
    members.setAttribute("country",
    members.getAttributeValue("country").replace("USA","USA␣CA"));
  }
  california.setAttribute("memberships", s.substring(0,s.length()-1));

  // put caldata into CAL
  List<Content> caldataelements =          // avoid ConcModException
     new LinkedList<Content>(caldata.getRootElement().getContent());
  for (Content el:caldataelements) el.detach();
  index = california.indexOf(california.getChild("city"));
  california.addContent(index,caldataelements);

  // put CAL country into mondial
  index = mondialEl.indexOf(usa);
  mondialEl.addContent(index,california);

  // change coountry attr and delete province attr for CAL cities
  for (Element el: (List<Element>)california.getChildren("city")) {
    el.setAttribute("country","CA");
    el.removeAttribute("province");
  }
  // add is_country_cap to the capital
  // xpath = xpf.compile("id(@capital)"); // no id function in jdom XPath!
  xpath = xpf.compile("city[@id=parent::*/@capital]");
  Element capitalcity = (Element)xpath.evaluateFirst(california);
  capitalcity.setAttribute("is_country_cap","yes");

  // borders of neighbors and USA-neighbors-borders
  for (Element el: (List<Element>)california.getChildren("border")) {
    String country = el.getAttributeValue("country");
    Double len = new Double(el.getAttributeValue("length"));
```

```
  if (!"USA".equals(country)) {
    xpath = xpf.compile("//country[@car_code='" + country + "']");
    Element countryEl = (Element)xpath.evaluateFirst(mondial);
    xpath = xpf.compile("border[@country='USA']");
    Element borderEl = (Element)xpath.evaluateFirst(countryEl);
    Double oldborderlength =
    new Double(borderEl.getAttributeValue("length"));
    Double newborderlength = oldborderlength - len;
    borderEl.setAttribute("length", newborderlength.toString());
    index = countryEl.indexOf(borderEl);
    countryEl.addContent(index,
    new Element("border").setAttribute("country","CA")
    .setAttribute("length",len.toString()));

    xpath = xpf.compile("border[@country='" + country + "']");
    Element usaborder     = (Element)xpath.evaluateFirst(usa);
    usaborder.setAttribute("length", newborderlength.toString());
  }
  else {
    index = usa.indexOf(usa.getChild("border"));
    usa.addContent(index,
    new Element("border").setAttribute("country","CA")
    .setAttribute("length",len.toString()));
  }
}

// geo things located in CAL
xpath = xpf.compile("//*[located/@country␣=␣'USA']");
List<Element> geos = xpath.evaluate(mondial);
for (Element geo:geos) {
  xpath = xpf.compile("located[@country␣=␣'USA']");
  Element locEl = (Element)xpath.evaluateFirst(geo);
  String provs = locEl.getAttributeValue("province");
  boolean inCal = false;
  String newprovs = "";
  for (String prov : provs.split("␣")) {
    if (calID.equals(prov)) inCal = true;
    else newprovs += prov + "␣";
  }
  if ("".equals(newprovs)) { // not in remaining USA
    locEl.detach();
    geo.setAttribute("country",
    geo.getAttributeValue("country").replace("USA","CA"));
  }
  else if (inCal) { // and other US provs
    locEl.setAttribute("province",newprovs.substring(0,newprovs.length()-1));
    geo.setAttribute("country",
    geo.getAttributeValue("country").replace("USA","USA␣CA"));
  }
}

// details: update USA country data
// area attribute
usa.setAttribute("area",
  Float.toString(Float.parseFloat(usa.getAttributeValue("area")) - calarea));
Element encomp = (Element)usa.getChild("encompassed").clone();
index = california.indexOf(california.getChild("ethnicgroup"));
california.addContent(index, new Text("\n"));
california.addContent(index, encomp);
california.addContent(index, new Text("\n"));
california.addContent(index, new Element("government").setText("democracy"));
california.addContent(index, new Text("\n"));
california.addContent(index,
  new Element("indep_date").setAttribute("from","USA").setText("2016-12-01"));
```

```
    // populations     // avoid ConcModException
    List<Element> pops = new LinkedList<Element>(usa.getChildren("population"));
    for (Element popEl:pops) {
      int year = Integer.parseInt(popEl.getAttributeValue("year"));
      xpath = xpf.compile("population[@year='" + year + "']");
      Element calpopEl = (Element)xpath.evaluateFirst(california);
      if (calpopEl != null) {
        calpop = Integer.parseInt(calpopEl.getTextTrim()); // keep last one
        uspop = Integer.parseInt(popEl.getTextTrim());
        popEl.setText(Integer.toString(uspop - calpop));
      }
      else popEl.detach();
    }
    weighByPop("population_growth");
    weighByPop("infant_mortality");
    weighByPop("unemployment");

    // GDPs
    Element el = usa.getChild("gdp_total");
    usGDP = Integer.parseInt(el.getTextTrim());
    calGDP = Integer.parseInt(california.getChild("gdp_total").getTextTrim());
    el.setText(Float.toString(usGDP - calGDP));
    weighByGDP("gdp_agri");
    weighByGDP("gdp_ind");
    weighByGDP("gdp_serv");
    weighByGDP("inflation");

    handleByGroups("ethnicgroup");
    handleByGroups("religion");
    handleByGroups("language");

    // output
    mondial.setDocType(new DocType("mondial", "mondial.dtd"));
    XMLOutputter outputter = new XMLOutputter();
    outputter.output(mondial, System.out);
  } catch (Exception e) { e.printStackTrace(); }
}

private void weighByPop(String elementname) {
  Element el = usa.getChild(elementname);
  Float usvalue = Float.parseFloat(el.getTextTrim());
  Float calvalue =
        Float.parseFloat(california.getChild(elementname).getTextTrim());
  el.setText(Float.toString(
              (usvalue * uspop - calvalue * calpop) / (uspop - calpop)));
}
private void weighByGDP(String elementname) {
  Element el = usa.getChild(elementname);
  Float usvalue = Float.parseFloat(el.getTextTrim());
  Float calvalue =
        Float.parseFloat(california.getChild(elementname).getTextTrim());
  el.setText(Float.toString(
              (usvalue * usGDP - calvalue * calGDP) / (usGDP - calGDP)));
}
private void handleByGroups(String elementname) throws JDOMException {
  List<Element> uselems = usa.getChildren(elementname);
  for (Element uselem:uselems) {
    String name = uselem.getTextTrim();
    xpath = xpf.compile(elementname+"[.='"+name+"']/@percentage");
    Attribute calpercAt = (Attribute)xpath.evaluateFirst(california);
    if (calpercAt != null) {
      Float usvalue = Float.parseFloat(uselem.getAttributeValue("percentage"));
      Float calvalue = Float.parseFloat(calpercAt.getValue());
      uselem.setAttribute("percentage",
        Float.toString((usvalue * uspop - calvalue * calpop)
```

```
                          / (uspop - calpop)));
      }
    }
  }
  private String getTodayDate() {
    DateFormat format = new SimpleDateFormat();
    return format.format(new Date());
  }

  public static void main(String[] args) {
    try {
      CalexitDOM calexit =
        new CalexitDOM(new FileInputStream("mondial.xml"), System.out);
      calexit.doIt();
    } catch (FileNotFoundException e) { e.printStackTrace(); }
  }
}
```

Exercise 3.6 (Calexit in XSLT)
Solve the previous exercise with an XSLT transformation.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">
<xsl:output indent="yes"/>

<xsl:variable name="calData">
  <xsl:copy-of select="doc('caldata.xml')/country/*"/>
</xsl:variable>

<xsl:template name="insertCalData">
   <xsl:copy-of select="name|population"/>
   <xsl:copy-of select="$calData/(population_growth|infant_mortality|
          gdp_total|gdp_agri|gdp_ind|gdp_serv|inflation|unemployment)"/>
   <indep_date from="USA">2017-04-01</indep_date>
   <xsl:copy-of select="../government"/>
   <xsl:copy-of select="../encompassed"/>
   <xsl:copy-of select="$calData/(ethnicgroup|religion|language|border)"/>
</xsl:template>

<xsl:template name="updateUSData">
   <xsl:copy-of select="name"/>
   <xsl:apply-templates
        select="population[@year=../province[name='California']/population/@year]"/>
   <xsl:apply-templates select="population_growth|infant_mortality|
          gdp_total|gdp_agri|gdp_ind|gdp_serv|inflation|unemployment"/>
   <xsl:copy-of select="government"/>
   <xsl:copy-of select="encompassed"/>
   <xsl:apply-templates select="ethnicgroup|religion|language"/>
   <xsl:apply-templates select="border"/>
   <border country="CA" length="{$calData/border[@country='USA']/@length}"/>
</xsl:template>

<xsl:template match="population[../@car_code='USA']">
  <population measured="{@measured}" year="{@year}">
      <xsl:value-of
       select=". - ../province[name='California']/population[@year=current()/@year]"/>
  </population>
</xsl:template>

<!-- note:    "(a|b|c)[...]" is not allowed in XSLT patterns -->
<xsl:template match="population_growth[../@car_code='USA']
                   | infant_mortality[../@car_code='USA']
                   | unemployment[../@car_code='USA']">
```

```
  <!-- weighted by population; calculate rest without California -->
  <xsl:element name="{name()}">
    <xsl:value-of
        select="(. * ../population[last()] - $calData/*[name()=current()/name()] *
          ../province[name='California']/population[last()]) div
                (../population[last()] -
                 ../province[name='California']/population[last()])"/>
  </xsl:element>
</xsl:template>

<xsl:template match="gdp_total[../@car_code='USA']">
  <xsl:element name="{name()}">
    <xsl:value-of select=". - $calData/*[name()=current()/name()]"/>
  </xsl:element>
</xsl:template>

<xsl:template match="gdp_agri[../@car_code='USA']|gdp_ind[../@car_code='USA']
      |gdp_serv[../@car_code='USA']|inflation[../@car_code='USA']">
  <!-- weighted by GDPtotal -->
  <xsl:element name="{name()}">
    <xsl:value-of
        select="(. * ../gdp_total -
                   $calData/*[name()=current()/name()] * $calData/gdp_total)
                div (../gdp_total - $calData/gdp_total) "/>
  </xsl:element>
</xsl:template>

<xsl:template match="ethnicgroup[../@car_code='USA']
                     | religion[../@car_code='USA']
                     | language[../@car_code='USA']">
  <!-- weighted by population; calculate rest without California -->
  <xsl:element name="{name()}">
    <xsl:attribute name="percentage">
    <xsl:value-of
select=" (@percentage * ../population[last()]
                 - $calData/*[name()=current()/name()
                              and text()=current()/text()]/@percentage *
                  ../province[name='California']/population[last()]) div
           (../population[last()] - ../province[name='California']/population[last()])"/>
    </xsl:attribute>
    <xsl:value-of select="text()"/>
  </xsl:element>
</xsl:template>

<!-- borders between USA and neighbors of CA change their length (USA-MEX) -->
<xsl:template match="border[../@car_code='USA' and @country=$calData/border/@country]">
  <!-- subtract CAL border portion if exists, otherwise match=* will apply and copy unchanged -->
    <xsl:element name="{name()}">
    <xsl:attribute name="country" select="@country"/>
    <xsl:attribute name="length">
      <xsl:value-of
    select="@length - $calData/*[name()=current()/name()
                                 and @country=current()/@country]/@length"/>
    </xsl:attribute>
  </xsl:element>
</xsl:template>

<!-- borders between neighbors of CA and USA/CA see below for generic pattern -->

<!-- ####### ... and now to the generic, structural updates ... ####################### -->

<xsl:template match="*">
  <xsl:element name="{name()}">
    <xsl:copy-of select="@*"/>
    <xsl:apply-templates select="*|text()"/>
```

```
  </xsl:element>
</xsl:template>

<xsl:template match="country[@car_code='USA']">
   <country>
     <xsl:copy-of select="@*[name()!='area']"/>
     <xsl:attribute name="area" select="@area - province[name='California']/area"/>
     <xsl:call-template name="updateUSData"/>
     <xsl:apply-templates select="province[name!='California']"/>
   </country>
   <xsl:apply-templates select="province[name='California']"/>
</xsl:template>

<xsl:template match="province[@country='USA' and name='California']">
   <country car_code="CA" area="{area}">
     <xsl:copy-of select="@capital"/>
     <xsl:attribute name="memberships">
      <xsl:value-of select="id(../@memberships)[not (abbrev=('G-5','G-7'))]/abbrev"/>
     </xsl:attribute>
     <xsl:call-template name="insertCalData"/>
     <xsl:apply-templates select="city"/>
   </country>
</xsl:template>

<xsl:template match="city[@country='USA' and id(@province)/name='California']">
   <city id="{@id}" country="CA">
     <xsl:if test="@id=../@capital">
       <xsl:attribute name="is_country_cap">yes</xsl:attribute>
     </xsl:if>
     <xsl:apply-templates select="*"/>
   </city>
</xsl:template>

<!-- borders between neighbors (MEX) of CA and USA change their length (MEX-USA)
     and add MEX-CA border-->
<xsl:template match="border[../@car_code=$calData/border/@country and @country='USA']">
  <border country="{@country}"
    length="{@length - $calData/*[name()=current()/name() and
                                  @country=current()/../@car_code]/@length}"/>
  <border country="CA"
     length="{$calData/*[name()=current()/name() and
                         @country=current()/../@car_code]/@length}"/>
</xsl:template>

<!-- geo things -->
<xsl:template match="*[located[@country='USA' and id(@province)/name='California']]">
  <xsl:variable name="usa">  <!-- to be selected in sequence for idrefs -->
    <xsl:if test="located[@country='USA' and id(@province)/name!='California']">
      <xsl:text>USA</xsl:text>
    </xsl:if>
  </xsl:variable>
  <xsl:element name="{name()}">
    <xsl:copy-of select="@*[not (name()='country')]"/>
    <xsl:attribute name="country">
      <!-- the ... or @car_code=$usa ... is a tricky way to deal with the whitespace separator
   for the generated ID list -->
      <xsl:value-of
       select="id(@country)[not (@car_code='USA') or @car_code=$usa]/@car_code, 'CA'"/>
    </xsl:attribute>
    <xsl:apply-templates select="*"/>
  </xsl:element>
</xsl:template>

<xsl:template match="located[@country='USA' and
             id(@province)/name='California' and count(id(@province))=1]">
```

```
  <!-- nothing, since CA does not have provinces -->
</xsl:template>

<xsl:template match="located[@country='USA' and
             id(@province)/name='California' and count(id(@province))>1]">
  <located country='USA'>
     <xsl:attribute name="province">
       <xsl:value-of select="id(@province)[name!='California']/@id"/>
     </xsl:attribute>
  </located>
</xsl:template>

<xsl:template match="members[not (../name=('G-5','G-7')) and id(@country)/@car_code='USA']">
  <members type="{@type}">
     <xsl:attribute name="country" select="id(@country)/@car_code, 'CA'"/>
     <xsl:apply-templates select="*"/>
  </members>
</xsl:template>

<xsl:template match="text()">
  <xsl:value-of select="."/>
</xsl:template>
</xsl:stylesheet>
```

Exercise 3.7 (Calexit in SAX/StAX)

Solve the previous exercise with a SAX or StAX transformation, again writing the result out as XML into a file.

- Roughly, 4 tasks:
  - make California a country (and put it anywhere in Mondial)
  - adapt the USA country data (area, population etc.)
  - adapt the border lengths and data of its neigbors,
  - adapt all @country and <located> data of geographical things. Here, the @country is in the opening tag, and the <located> follows later. Thus, some buffering is needed.
- For processing the USA country data (weighing population growth, languages etc. by population), the most recent California population is needed.
  Thus, the first part of the USA <country> element must be buffered anyway

⇒ Output California *before* the USA. Then, only the USA country data and its first provinces must be buffered.
(most people intuitively put it behind it, which requires buffering the first part of the USA, finding California, then the first part of USA can be done, California must be buffered, the remaining part of the USA can be directly piped into the output, and then California must be output).

- Buffering: (at least) three alternatives:
  - build a local DOM,
  - StAX it into an XML file, read this file again with another reader,
  - StAX it into a BufferedOutputStream over an XMLStream, and flush this one into the original output stream (for the unchanged first US provinces+cities), or StAX-process the resulting PipedInputStream (for the US country data and for the geos).
- Dealing time vs. space, there is also a 2-Pass-Approach:
  - first, read through mondial.xml, considering only California, generate its country output (California from mondial, caldata.xml, and some information from USA (government, en-

compasses)) and write it already into the result stream. Keep relevant information (population) from it to adapt the USA country data.
  - in this run, also a small data structure can be build, which geo things in the USA are only USA, only CAL, or both.
  - second run: all countries (unchanged except borders), including USA, adapt USA geo things accordingly.
- Below, first a One-Pass solution is shown that plays also a bit with BufferedStreams:
  It processes all countries until the USA start (most of them unchanged, just for CA's neighbors, the borders ara adapted), then it buffers the beginning of the USA (country data and some provinces). Then it finds California, and processes it (inserting the `caldata.xml` country data). Then, it takes the buffered USA portion, processes its country data (removing the area, population, GDP etc.) from California, outputs the buffered provinces and then continues with the remaining provinces. Then it processes the remaining countries.
  Then, it processes the organizations, and finally the geo things. Here it must again buffer the first facts of each fo them, until it knows from the <`located`>-elements whether the `@country` attribute must be changed.
- Additionally, a Two-Pass solution is shown, based on the proceeding of the One-Pass variant. In the first pass, it processes all non-neighbor countries until it reaches the USA. Then, it ignores the USA and processes California as above (keeping relevant data from `caldata.xml` in memory for the second pass). Afterwards, it just scans all geo objects wnd stores which of them are located in (i) only California, and (ii) USA and California.
  In the second pass, it continues with all neighbor countries of the USA, the USA, and all countries after the USA. Then, it processes the organizations and the geo things.
- Third, a variant of the one-pass solution is given that does not store the buffered data in an XMLStream, but in a DOM. It is written there by deviating the output in to a stream that feeds a SAXBuilder to build a JDOM (i.e., the provonces of the USA before California, and later the beginnings of the geo things up to the USA-located element). Then, the appropriate updates are applied (rather copied from the JDOM solution). When the (modified) DOM is written back into the main output, there is the issue to "kill" its closing element (because further unchanged contents of the modified element follows, i.e., the remaining USA provinces, and the rest of the geo things). For this, it is not directly output into the final output, but runs through another StAX copying that remves the final closing tag.

```
import java.io.InputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.OutputStream;
import java.io.PipedInputStream;
import java.io.PipedOutputStream;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;
import java.util.LinkedList;
import java.util.Map;
import java.util.HashMap;

import java.io.PipedInputStream;
import java.io.PipedOutputStream;
import java.io.BufferedOutputStream;

import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLOutputFactory;
import javax.xml.stream.XMLStreamConstants;
```

```
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamReader;
import javax.xml.stream.XMLStreamWriter;

import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.Attribute;
import org.jdom2.Text;
import org.jdom2.Namespace;
import org.jdom2.input.SAXBuilder;
import org.jdom2.xpath.XPathFactory;
import org.jdom2.xpath.XPathExpression;
import org.jdom2.filter.Filters;

public class CalexitStAXOnePass {

  FileInputStream inputStream;
  OutputStream outputStream;
  XMLInputFactory inputFactory = XMLInputFactory.newInstance();
  Document caldata = null;
  List<Attribute> neighborcodes = new LinkedList<Attribute>();  // empty
  Float calGDP = null;
  Float usGDP = null;
  Integer calpop = null;
  Integer uspop = null;
  String usmemberships = null;
  String calAreaS = null;
  boolean useBufferedStream = true;  // false: write CAL to file,
                                     // true:  BufferedOutputStream
  boolean geoOnlyCAL = false;
  boolean geoOnlyUSA = false;
  boolean geoCALUSA = false;
  List<String> calpopyears = new LinkedList<String>();
  List<String> calpops = new LinkedList<String>();

  XPathFactory xpf = XPathFactory.instance();
  XPathExpression xpath = null; // only for multiple short term use

  public enum tasks { unchanged, handleUSdata, handleCalCountry,
                      handleLocateds, handleCalData };

  public CalexitStAXOnePass(FileInputStream in, OutputStream out) {
    this.inputStream = in;
    this.outputStream = out;
  }

  public void startParsing() {
    SAXBuilder builder = new SAXBuilder();
    File xmlFile = new File("caldata.xml");
    try {
      caldata = (Document) builder.build(xmlFile);
      xpath = xpf.compile("./country/border/@country");
      neighborcodes = xpath.evaluate(caldata); // includes "USA"
      xpath = xpf.compile("./country/gdp_total/text()");
      calGDP = new Float(((Text)(xpath.evaluateFirst(caldata))).getTextTrim());

      XMLOutputFactory outputFactory = XMLOutputFactory.newInstance();
      XMLStreamReader mparser = inputFactory.createXMLStreamReader(inputStream);
      XMLStreamWriter mwriter = outputFactory.createXMLStreamWriter(outputStream);
      XMLStreamReader caldataparser =
        inputFactory.createXMLStreamReader(new FileInputStream("caldata.xml"));
      boolean goOn = true;
      boolean inusa = false;
      boolean inusprovince = false;
      boolean waitforprovname = false;
```

```
boolean incalifornia = false;
boolean incalcity = false;
boolean calfirstcityover = false;
String calID = null;
String inneighborcountry = null;
boolean inorganization = false;
String orgname = null;
boolean waitforUSAlocated = false;
String provID = "";
String capital = "";
XMLStreamWriter usawriter = null;

// store first USA provinces+cities intermediately in BufferedStreams
// that is then directly flushed into the final output:
BufferedOutputStream usaBOS = null;

// write US Geo things intermediately in a Buffered stream while
// waiting for "located"   (also used for US country data before)
PipedOutputStream geoPOS = null;
BufferedOutputStream geoBOS = null;
PipedInputStream geoPIS = null;
XMLStreamReader geoparser = null;

// The following is first written into temporary files/streams:
// USA country data (to be updated later when CAL population is known
// for weighing infant mortality, religions/languages/ethnicgroups)
// and its provinces occurring before CAL.
// California is directly written as a new country to mondial,
// then the buffered USA data is adapted/flushed an the rest is
// directly written to mondial.
if (!useBufferedStream) { // write into a file
  usawriter =
    outputFactory.createXMLStreamWriter(new FileOutputStream("usa.xml"));
} else {
    // write into a buffered Stream with is later
    // flushed directly into outputStream
    usaBOS = new BufferedOutputStream(outputStream,1000000);
    usawriter = outputFactory.createXMLStreamWriter(usaBOS);
}
// for geos always use BufferedStreams:
geoPOS = new PipedOutputStream();
geoBOS = new BufferedOutputStream(geoPOS,100000); // write tmp BOS->POS->PIS
geoPIS = new PipedInputStream();
geoPIS.connect(geoPOS);  // <- this will be read
XMLStreamWriter tmpgeowriter = outputFactory.createXMLStreamWriter(geoBOS);

XMLStreamReader parser = mparser;
XMLStreamWriter writer = mwriter;
writer.writeStartDocument("UTF-8", "1.0");
writer.writeCharacters("\n");
// writer.writeDTD("<!DOCTYPE mondial SYSTEM 'mondial.dtd'>");
// writer.writeCharacters("\n");

while (goOn) {
  int event = parser.next();
  switch(event) {
  case XMLStreamConstants.DTD:
    String doctypedecl = parser.getText();
    writer.writeDTD(doctypedecl);
    writer.writeCharacters("\n");
    break;
  case XMLStreamConstants.END_DOCUMENT:
    parser.close();
    writer.flush();
    writer.close();
```

```
        goOn = false;
        break;
      case XMLStreamConstants.START_ELEMENT:
        if ("country".equals(parser.getLocalName())) {
          String code = parser.getAttributeValue(null, "car_code");
          if ("USA".equals(code)) {
            inusa = true;
            usmemberships = parser.getAttributeValue(null, "memberships");
            writer = tmpgeowriter;
          }
          else if (!neighborcodes.isEmpty()) {
            for (int i=0; i< neighborcodes.size(); i++)
              if (((Attribute)(neighborcodes.get(i))).getValue().equals(code))
                inneighborcountry = code;
          }
          copyStartElement(parser,writer); writer.writeCharacters("\n");
          break;
        }
        else if (inusa && !inusprovince // all elements between <pop> and first <province>
                 && !"population".equals(parser.getLocalName())
                 && !"province".equals(parser.getLocalName())) {
          copyStartElement(parser,writer); // writer is tmpgeowriter;
              // change all of them later
          break;
        }
        else if (inusa && "province".equals(parser.getLocalName())) {
          if (writer == tmpgeowriter) { // now first province in USA
            writer.writeStartElement("stop");
            writer.writeEndElement();
            writer = usawriter;  // switch from tmpgeowriter to usawriter
          }
          provID = parser.getAttributeValue(null, "id");
          capital = parser.getAttributeValue(null, "capital");
          inusprovince = true;
          waitforprovname = true;
          break;
        }
        else if (inusprovince && waitforprovname
                 && "name".equals(parser.getLocalName())) {
          waitforprovname = false;
          String provname = parser.getElementText();
          if ("California".equals(provname)) {
            incalifornia = true;
            writer = mwriter;
            calID = provID;
            writer.writeStartElement("country");
            writer.writeAttribute("car_code","CA");
            writer.writeAttribute("capital",capital);
            writer.writeAttribute("memberships",  // dirty. But I don't want to wait for the orgs.
                    usmemberships.replace("␣org-G-5","").replace("␣org-G-7",""));
          }
          else {
            writer.writeStartElement("province");
            writer.writeAttribute("id",provID);
            writer.writeAttribute("country","USA");
            writer.writeAttribute("capital",capital); writer.writeCharacters("\n");
            writer.writeStartElement("name");
            writer.writeCharacters(provname);
            writer.writeEndElement(); writer.writeCharacters("\n");
          }
          break;
        }
        else if (incalifornia && "area".equals(parser.getLocalName())) {
          calAreaS = parser.getElementText();
          writer.writeAttribute("area",calAreaS);
```

```
  writer.writeCharacters("\n");
  writer.writeStartElement("name");
  writer.writeCharacters("California");
  writer.writeEndElement(); writer.writeCharacters("\n");
  break;
}
else if (incalifornia && !incalcity
         && "population".equals(parser.getLocalName())) {
  copyStartElement(parser,writer);
  String year = parser.getAttributeValue(null, "year");
  calpopyears.add(year);
  parser.next();  // text content
  String calpopS = parser.getText();
  writer.writeCharacters(calpopS);
  calpops.add(calpopS);
  calpop = new Integer(calpopS); // note: last one stays as calpop
  parser.next(); // </population>
  writer.writeEndElement(); writer.writeCharacters("\n");
  break;
}
else if (incalifornia && "city".equals(parser.getLocalName())) {
  if (!calfirstcityover) {
    calfirstcityover = true;
    copyFromXMLStream(caldataparser, writer, tasks.handleCalData);
  }
  incalcity = true;
  writer.writeStartElement("city");
  String cityid = parser.getAttributeValue(null,"id");
  writer.writeAttribute("id",cityid);
  writer.writeAttribute("country","CA");
  if (cityid.equals(capital))
      writer.writeAttribute("is_country_cap","yes");
  writer.writeCharacters("\n");
  break;
}
else if (inneighborcountry != null
         && "border".equals(parser.getLocalName())) {
  String tocountry = parser.getAttributeValue(null,"country");
  if ("USA".equals(tocountry)) { // border MEX to USA
    writer.writeStartElement("border");
    Float oldlength =
        new Float(parser.getAttributeValue(null,"length"));
    Map<String,Object> vars = new HashMap<String,Object>();
    vars.put("neighbor",null);
    xpath = xpf.compile("./country/border[@country=$neighbor]/@length",
                        Filters.attribute(), vars, (Namespace[]) null);
    xpath.setVariable("neighbor", inneighborcountry);
    Attribute callengthAttr =
        (Attribute) xpath.evaluateFirst(caldata);
    Float callength = new Float(callengthAttr.getValue());
    writer.writeAttribute("country","USA");
    writer.writeAttribute("length",
                          Float.toString(oldlength-callength));
    writer.writeEndElement();writer.writeCharacters("\n");
    // add border to CA:
    writer.writeEmptyElement("border");
    writer.writeAttribute("country","CA");
    writer.writeAttribute("length", callength.toString());
    parser.next(); // ignore this EndElem because <border/> is an empty el.
    writer.writeCharacters("\n");
  }
  else copyStartElement(parser,writer);
  break;
}
// ######## and now to the organizations:
```

```
else if ("organization".equals(parser.getLocalName())) {
  inorganization = true;
  copyStartElement(parser,writer);writer.writeCharacters("\n");
  break;
}
else if (inorganization && "name".equals(parser.getLocalName())) {
  copyStartElement(parser,writer);
  parser.next();
  orgname = parser.getText();
  writer.writeCharacters(orgname);
  break;
}
else if (inorganization && "members".equals(parser.getLocalName())) {
  String type = parser.getAttributeValue(null, "type");
  String countries = parser.getAttributeValue(null, "country");
  if (!orgname.equals("Group␣of␣5") && !orgname.equals("Group␣of␣7")
      && countries.contains("USA")) { // add CA to the members
      writer.writeStartElement("members");
      writer.writeAttribute("type","type");
      writer.writeAttribute("country",
                            countries.replace("USA","USA␣CA"));
  }
  else
      copyStartElement(parser,writer);
  break;
}
// ######## and now to the geo things:
// disadvantage of StAX against XSLT: all geos must explicitly be listed
// even worse disadvantage: read down to "located" to know whether
//       located in CAL
// => use again the tmp BufferedStream
else if ("river".equals(parser.getLocalName())
     || "lake".equals(parser.getLocalName())
     || "sea".equals(parser.getLocalName())
     || "source".equals(parser.getLocalName())
     || "estuary".equals(parser.getLocalName())
     || "island".equals(parser.getLocalName())
     || "mountain".equals(parser.getLocalName())
     || "desert".equals(parser.getLocalName())) {
  for (String code : parser.getAttributeValue(null, "country").split("␣"))
    if ("USA".equals(code)) {
      waitforUSAlocated = true;
      writer = tmpgeowriter;
    }
  copyStartElement(parser,writer); writer.writeCharacters("\n");
  break;
}
else if ("located".equals(parser.getLocalName())
         && waitforUSAlocated) {
  if ("USA".equals(parser.getAttributeValue(null, "country"))) {
    waitforUSAlocated = false;
    geoOnlyUSA = true;
    geoOnlyCAL = false;
    geoCALUSA = false;
    for (String locprovID :
         parser.getAttributeValue(null, "province").split("␣"))
      if (calID.equals(locprovID)) {
        if (parser.getAttributeValue(null, "province")
              .split("␣").length == 1)
            { geoOnlyCAL = true; geoOnlyUSA = false; }
        else  { geoCALUSA = true; geoOnlyUSA = false; }
      }

    // Now, we know whether the geo is in US/CAL
    // flush the tmp-geoBOS and process it to mondial/mwriter
```

```
      writer.writeStartElement("stop");
      writer.writeEndElement();
      final BufferedOutputStream geoBOS2 = geoBOS;
      new Thread() {
        public void run(){
          try{ geoBOS2.flush();
          } catch (Exception e) { e.printStackTrace(); }}}.start();
      Thread.sleep(100);
      writer = mwriter;  // from now on, write to mondial,
      // first the buffered contents ...
      copyFromXMLStream(geoparser, writer, tasks.handleLocateds);
      // then come back and write out this located
      if (geoOnlyUSA) {  // <located USA> unchanged
        copyStartElement(parser,writer);
        writer.writeEndElement(); writer.writeCharacters("\n");
      }
      else if (geoCALUSA) { // remove CA from provinces list
        writer.writeStartElement("located");
        writer.writeAttribute("country","USA");
        String newprovs = "";
        for (String locprovID :
             parser.getAttributeValue(null, "province").split("␣")) {
          if (!calID.equals(locprovID))
            newprovs = newprovs + locprovID + "␣";
        }
        newprovs = newprovs.substring(0,newprovs.length()-1);
        writer.writeAttribute("province",newprovs);
        writer.writeEndElement(); writer.writeCharacters("\n");
      }  // note: CAL has no province, thus no located needed for it
      parser.next();  // the end-element of the located
    }
    else // not the USA-located Element
        copyStartElement(parser,writer);
    break;
  }
  // all others: copy
  else copyStartElement(parser,writer);
    break;
  case XMLStreamConstants.CHARACTERS:
    writer.writeCharacters(parser.getText());
    break;
  case XMLStreamConstants.END_ELEMENT:
    if (incalcity && "city".equals(parser.getLocalName())) {
      incalcity = false;
      writer.writeEndElement(); writer.writeCharacters("\n");
    break;
    }
    else if (incalifornia && "province".equals(parser.getLocalName())) {
      incalifornia = false;
      // end CAL's country element
      writer.writeEndElement(); writer.writeCharacters("\n");
      writer = mwriter; // write the following again to mondial

      // Get USA country data from tmpgeo buffered Stream:
      //   opening <country ... > tag (update area) + <name>...</>
      final BufferedOutputStream geoBOS2 = geoBOS;
      new Thread() {
        public void run(){
          try{ geoBOS2.flush();
             } catch (Exception e) { e.printStackTrace(); }}}.start();
      Thread.sleep(100);
      if (geoparser == null)
        // this must not be done before there is something to parse
        geoparser = inputFactory.createXMLStreamReader(geoPIS);
      geoparser.next(); // <country USA opening>
```

```
            writer.writeStartElement("country");
            writer.writeAttribute("car_code",
                      geoparser.getAttributeValue(null, "car_code"));
            Integer newarea =
                new Integer(geoparser.getAttributeValue(null, "area")) -
                new Integer(calAreaS);
            writer.writeAttribute("area", newarea.toString());
            writer.writeAttribute("capital",
                      geoparser.getAttributeValue(null, "capital"));
            writer.writeAttribute("memberships",
                      geoparser.getAttributeValue(null, "memberships"));
            geoparser.next(); // whitespace CR after closing <country> before <name>
            writer.writeCharacters("\n");
            geoparser.next();
            writer.writeStartElement(geoparser.getLocalName()); // <name>
            geoparser.next();
            writer.writeCharacters(geoparser.getText()); // United States
            geoparser.next();
            writer.writeEndElement();
            copyFromXMLStream(geoparser, writer, tasks.handleUSdata);
            // after this, add border USA to CAL
            xpath = xpf.compile("./country/border[@country␣=␣'USA']/@length");
            String callength =
               ((Attribute)(xpath.evaluateFirst(caldata))).getValue();
            writer.writeEmptyElement("border");
            writer.writeAttribute("country", "CA");
            writer.writeAttribute("length", callength);
            // writer.writeEndElement(); // no EndElem for EmptyElem <border/>
            writer.writeCharacters("\n");
            // ... and write out USABOS with cities and provinces (everything that
            //   occurred in mondial.xml before California) to writer
            if (!useBufferedStream) { ;
              XMLStreamReader usafileparser =
                inputFactory.createXMLStreamReader(new FileInputStream("usa.xml"));
              copyFromXMLStream(usafileparser, writer, tasks.unchanged);
            } else { // BufStream
                usaBOS.flush();
            }
            break;
          }
          else if (inusprovince && "province".equals(parser.getLocalName())) {
            inusprovince = false;
            writer.writeEndElement(); writer.writeCharacters("\n");
            break;
          }
          else if ("organization".equals(parser.getLocalName())) {
            inorganization = false;
            writer.writeEndElement(); writer.writeCharacters("\n");
            break;
          }
          else if (inusa && "country".equals(parser.getLocalName())) {
            inusa = false;
              // no "break" ...
          }
          else if ("country".equals(parser.getLocalName())) {
            inneighborcountry = null;
            // no "break" ...
          }
          // in all cases that did not have a "break": close Element
          writer.writeEndElement();
          writer.writeCharacters("\n");
          break;
        }
      }
    } catch (Exception e) { e.printStackTrace(); }
```

```
}

private void copyStartElement(XMLStreamReader parser, XMLStreamWriter writer)
    throws XMLStreamException {
  writer.writeStartElement(parser.getLocalName());
  if (parser.getAttributeCount() > 0) {
    int n = parser.getAttributeCount();
    for (int i = 0; i < parser.getAttributeCount(); i++)
      writer.writeAttribute(parser.getAttributeLocalName(i),
                            parser.getAttributeValue(i));
  }
}

private void copyFromXMLStream(XMLStreamReader parser,
                               XMLStreamWriter writer, tasks task)
    throws Exception {
  boolean goOn = true;
  // System.out.println("copyFromXMLStream");
  while (goOn) {
    int event = parser.next();
    switch (event) {
    case XMLStreamConstants.START_ELEMENT:
      if ((task == tasks.handleLocateds || task == tasks.handleUSdata) &&
          "stop".equals(parser.getLocalName())) {
          break; // ignore; react on its END_ELEMENT
      }
      else if (task == tasks.handleLocateds &&
         ("river".equals(parser.getLocalName())
          || "lake".equals(parser.getLocalName())
          || "sea".equals(parser.getLocalName())
          || "source".equals(parser.getLocalName())
          || "estuary".equals(parser.getLocalName())
          || "island".equals(parser.getLocalName())
          || "mountain".equals(parser.getLocalName())
          || "desert".equals(parser.getLocalName()))) {
        String countries = parser.getAttributeValue(null, "country");
        if (geoOnlyUSA)
            copyStartElement(parser,writer);
        else { // geoOnlyCAL or geoCALUSA
          writer.writeStartElement(parser.getLocalName());
          int n = parser.getAttributeCount();
          for (int i = 0; i < parser.getAttributeCount(); i++) {
            if (!"country".equals(parser.getAttributeLocalName(i)))
              writer.writeAttribute(parser.getAttributeLocalName(i),
                                    parser.getAttributeValue(i));
            else {
              if (geoOnlyCAL)
                  countries = countries.replace("USA","CA");
              else if (geoCALUSA)
                  countries = countries.replace("USA","USA␣CA");
              writer.writeAttribute("country", countries);
            }
          }
        }
        break;
      }
      // US country data: update wrt. CAL data (uses calpop)
      else if (task == tasks.handleUSdata) {
        if ("population".equals(parser.getLocalName())) {
          String year = parser.getAttributeValue(null, "year");
          parser.next();  // text content
          uspop = new Integer(parser.getText());
          // note: last one stays as uspop
          parser.next();  // endElement
          boolean found = false;
```

```
  for (int i=0; i < calpops.size(); i++) {
    if (calpopyears.get(i).equals(year)) {
      found = true;
      calpop = new Integer(calpops.get(i));
      writer.writeStartElement("population");
      writer.writeAttribute("year",year);
      writer.writeCharacters(new Integer(uspop - calpop).toString());
      writer.writeEndElement();
    }
  }
  if (!found) parser.next(); // read the linebreak for omitted <pop>
  break;
}
else if ("government".equals(parser.getLocalName()) ||
         "indep_date".equals(parser.getLocalName())) {
  copyStartElement(parser,writer);
  parser.next();
  writer.writeCharacters(parser.getText());
  parser.next(); // end element
  writer.writeEndElement();
  break;
}
else if ("encompassed".equals(parser.getLocalName()))  {
  copyStartElement(parser,writer);
  parser.next(); // end element
  writer.writeEndElement();
  break;
}
else if (parser.getAttributeCount() == 0) {
    /* population_growth?,infant_mortality?,
       gdp_total?,gdp_agri?,gdp_ind?,gdp_serv?,inflation?,unemployment?*/
  String elementname = parser.getLocalName();
  writer.writeStartElement(elementname);
  parser.next();
  Float usvalue = new Float(parser.getText());
  xpath = xpf.compile("./country/" + elementname + "/text()");
  Float calvalue =
      new Float(((Text)(xpath.evaluateFirst(caldata))).getTextTrim());
  if ("population_growth".equals(elementname)
      || "infant_mortality".equals(elementname)
      || "unemployment".equals(elementname)) {
    writer.writeCharacters(new Float((usvalue * uspop - calvalue * calpop)
                                     / (uspop - calpop)).toString());
  }
  else if ("gdp_total".equals(elementname)) {
    usGDP = usvalue;
    writer.writeCharacters(new Float(usvalue - calvalue).toString());
  }
  else if ("gdp_agri".equals(elementname)
           || "gdp_ind".equals(elementname)
           || "gdp_serv".equals(elementname)
           || "inflation".equals(elementname))  {
    writer.writeCharacters(new Float((usvalue * usGDP - calvalue * calGDP)
                                     / (usGDP - calGDP)).toString());
  }
  parser.next(); // end element
  writer.writeEndElement();
  break;
}
else if ("ethnicgroup".equals(parser.getLocalName())
         || "religion".equals(parser.getLocalName())
         || "language".equals(parser.getLocalName()))  {
  String elementname = parser.getLocalName();
  Float oldperc = new Float(parser.getAttributeValue(null,"percentage"));
  parser.next();
```

```
        String name = parser.getText();
        xpath = xpf.compile("./country/" + elementname
                            + "[.='" + name + "']/@percentage");
        Attribute attr = (Attribute)(xpath.evaluateFirst(caldata));
        Double calperc = 0.0;
        if (attr != null) calperc = new Double(attr.getValue());
        writer.writeStartElement(elementname);
        writer.writeAttribute("percentage",
                              new Double((oldperc * uspop - calperc * calpop)
                                         / (uspop - calpop)).toString());
        writer.writeCharacters(name);
        writer.writeEndElement();
        parser.next(); // end element
        break;
      }
      else if ("border".equals(parser.getLocalName())) {
        String country = parser.getAttributeValue(null,"country");
        Double length = new Double(parser.getAttributeValue(null,"length"));
        xpath = xpf.compile(
                "./country/border[@country = '" + country + "']/@length");
        Attribute attr = (Attribute)(xpath.evaluateFirst(caldata));
        Double callength = 0.0;
        if (attr != null) callength = new Double(attr.getValue());
        writer.writeEmptyElement("border");
        writer.writeAttribute("country", country);
        writer.writeAttribute("length",
                  new Float(length - callength).toString());
        parser.next(); // ignore this EndElement because <border .../>
                       // is an empty element
      }
      break;
    }
    if (task == tasks.handleCalData
        && "country".equals(parser.getLocalName())) { // in caldata.xml
      parser.next(); // ignore a "/n"
      break; // ignore;
    }
    copyStartElement(parser, writer);
    break;
  case XMLStreamConstants.CHARACTERS:
    writer.writeCharacters(parser.getText());
    break;
  case XMLStreamConstants.END_ELEMENT:
    if ((task == tasks.handleLocateds || task == tasks.handleUSdata)
        && "stop".equals(parser.getLocalName())) {
        // System.out.println("CopyFromStream: stop found");
      goOn = false;
      break;
    }
    else if (task == tasks.handleCalData
             && "unemployment".equals(parser.getLocalName())) {
      // add independence, government and encompassed behind unemployment
      writer.writeEndElement(); writer.writeCharacters("\n");
      writer.writeStartElement("indep_date");
      writer.writeAttribute("from", "USA");
      writer.writeCharacters(getTodayDate());
      writer.writeEndElement(); writer.writeCharacters("\n");
      writer.writeStartElement("government");
      writer.writeCharacters("democracy");
      writer.writeEndElement(); writer.writeCharacters("\n");
      writer.writeStartElement("encompassed");
      writer.writeAttribute("continent", "america");
      writer.writeAttribute("percentage", "100");
      writer.writeEndElement(); // no \n here necessary
      break;
```

```
        }
        else if (task == tasks.handleCalData
                 && "country".equals(parser.getLocalName())) { // in caldata.xml
          goOn = false;
          break;
        }
        else if (task == tasks.handleCalCountry
                 && "country".equals(parser.getLocalName()))
          goOn = false;
        writer.writeEndElement();
      }
    }
  }

  private String getTodayDate() {
    DateFormat format = new SimpleDateFormat();
    return format.format(new Date());   }

  public static void main(String[] args) {
    try {
      CalexitStAXOnePass calexit =
          new CalexitStAXOnePass(new FileInputStream("mondial.xml"), System.out);
      calexit.startParsing();
    } catch (Exception e) { e.printStackTrace(); }
 }}
```

```
import java.io.InputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.OutputStream;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;
import java.util.LinkedList;
import java.util.Map;
import java.util.HashMap;

import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLOutputFactory;
import javax.xml.stream.XMLStreamConstants;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamReader;
import javax.xml.stream.XMLStreamWriter;

import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.Attribute;
import org.jdom2.Text;
import org.jdom2.Namespace;
import org.jdom2.input.SAXBuilder;
import org.jdom2.xpath.XPathFactory;
import org.jdom2.xpath.XPathExpression;
import org.jdom2.filter.Filters;

public class CalexitStAXTwoPass {

  String inputfile;
  OutputStream outputStream;
  XMLInputFactory inputFactory = XMLInputFactory.newInstance();
  XMLStreamReader parser;
  Document caldata = null;
  List<Attribute> neighborcodes = new LinkedList<Attribute>();  // empty
```

```
Float calGDP = null;
Float usGDP = null;
Integer calpop = null;
Integer uspop = null;
String usmemberships = null;
String calAreaS = null;

boolean geoOnlyCAL = false;
boolean geoCALUSA = false;
boolean beforeusa = false;
boolean afterusa = false;
String geoID = "";
List<String> calpopyears = new LinkedList<String>();
List<String> calpops = new LinkedList<String>();
List<String> calonlylocateds = new LinkedList<String>();
List<String> calusalocateds = new LinkedList<String>();

XPathFactory xpf = XPathFactory.instance();
XPathExpression xpath = null; // only for multiple short term use

public CalexitStAXTwoPass(String inputfile, OutputStream out) {
  this.inputfile = inputfile;
  this.outputStream = out;
}

public void startParsing() {
  SAXBuilder builder = new SAXBuilder();
  File xmlFile = new File("caldata.xml");

  try {
    caldata = (Document) builder.build(xmlFile);
    xpath = xpf.compile("./country/border/@country");
    neighborcodes = xpath.evaluate(caldata); // includes "USA"
    xpath = xpf.compile("./country/gdp_total/text()");
    calGDP = new Float(((Text)(xpath.evaluateFirst(caldata))).getTextTrim());

    XMLStreamReader mparser =
        inputFactory.createXMLStreamReader(new FileInputStream(inputfile));
    XMLStreamReader caldataparser =
        inputFactory.createXMLStreamReader(new FileInputStream("caldata.xml"));
    XMLOutputFactory outputFactory = XMLOutputFactory.newInstance();
    XMLStreamWriter writer = outputFactory.createXMLStreamWriter(outputStream);
    boolean goOn = true;
    boolean inusa = false;
    boolean inusprovince = false;
    boolean waitforprovname = false;
    boolean incalifornia = false;
    boolean incalcity = false;
    boolean calfirstcityover = false;
    boolean usacalborderadded = false;
    String calID = null;
    String inneighborcountry = null;
    boolean inorganization = false;
    String orgname = null;
    boolean waitforUSAlocated = false;
    String provID = "";
    String capital = "";

    // First Pass:
    // * all countries before USA which are not neighbors of CAL
    // * collect geo IDs located in CAL only or in USA and CAL
    parser = mparser; // read from mondial
    writer.writeStartDocument("UTF-8", "1.0");
    writer.writeCharacters("\n");
```

```
beforeusa = true;
while (goOn) {
  int event = parser.next();
  switch(event) {
    case XMLStreamConstants.DTD:
      String doctypedecl = parser.getText();
      writer.writeDTD(doctypedecl);
      writer.writeCharacters("\n");
      break;
    case XMLStreamConstants.END_DOCUMENT:
      parser.close();
      goOn = false;
      break;
    case XMLStreamConstants.START_ELEMENT:
      if (incalifornia && "country".equals(parser.getLocalName())) {
              // when parsing caldata
        parser.next(); // ignore a "/n"
        break;
      }
      else if (!incalifornia && beforeusa
               && "country".equals(parser.getLocalName())) {
        String code = parser.getAttributeValue(null, "car_code");
        if ("USA".equals(code)) {
          beforeusa = false;
          inusa = true;
          usmemberships = parser.getAttributeValue(null, "memberships");
        }
        else if (!neighborcodes.isEmpty()) {
          for (int i=0; i< neighborcodes.size(); i++)
          if (((Attribute)(neighborcodes.get(i))).getValue().equals(code))
          inneighborcountry = code;
        }
        if (beforeusa && inneighborcountry == null)
          copyStartElement(parser,writer);
        break;
      }
      else if (inusa && "province".equals(parser.getLocalName())) {
        provID = parser.getAttributeValue(null, "id");
        capital = parser.getAttributeValue(null, "capital");
        inusprovince = true;
        waitforprovname = true;
      }
      else if (inusprovince && waitforprovname
               && "name".equals(parser.getLocalName())) {
        waitforprovname = false;
        String provname = parser.getElementText(); // parser.next()
        if ("California".equals(provname)) {
          incalifornia = true;
          calID = provID;
          writer.writeStartElement("country");
          writer.writeAttribute("car_code","CA");
          writer.writeAttribute("capital",capital);
          writer.writeAttribute("memberships",
          usmemberships.replace("␣org-G-5","").replace("␣org-G-7",""));  // dirty.
          // But I don't want to wait for the orgs.names.
          // Could be done in another, preceding pass
        }  // not yet written the (California) name
        break; // note: first pass does not output US provinces
      }  // CAL area comes next, turn into an attribute, then write name ...
      else if (incalifornia && "area".equals(parser.getLocalName())) {
        calAreaS = parser.getElementText();
        writer.writeAttribute("area",calAreaS);
        writer.writeCharacters("\n");
        writer.writeStartElement("name");
        writer.writeCharacters("California");
```

```
  writer.writeEndElement(); writer.writeCharacters("\n");
  break;
}  // next: california's population elements:
else if (incalifornia && !incalcity
         && "population".equals(parser.getLocalName())) {
  copyStartElement(parser,writer);
  String year = parser.getAttributeValue(null, "year");
  calpopyears.add(year);
  parser.next();  // text content
  String calpopS = parser.getText();
  writer.writeCharacters(calpopS);
  calpops.add(calpopS);
  calpop = new Integer(calpopS); // note: last one stays as calpop
  parser.next(); // </population>
  writer.writeEndElement(); writer.writeCharacters("\n");
  break;
}
else if (incalifornia && "city".equals(parser.getLocalName())) {
  if (!calfirstcityover) {
    calfirstcityover = true;
    parser = caldataparser; // switch parser to read caldata first ;)
    parser.next(); // read caldata's START_DOCUMENT
    break;
  }
  else {
    incalcity = true;
    writer.writeStartElement("city");
    String cityid = parser.getAttributeValue(null,"id");
    writer.writeAttribute("id",cityid);
    writer.writeAttribute("country","CA");
    if (cityid.equals(capital))
    writer.writeAttribute("is_country_cap","yes");
    writer.writeCharacters("\n");
  }
  break;
}
// geos: put IDs into calonlylocateds/calusalocateds
else if ("river".equals(parser.getLocalName())
    || "lake".equals(parser.getLocalName())
    || "sea".equals(parser.getLocalName())
    || "source".equals(parser.getLocalName())
    || "estuary".equals(parser.getLocalName())
    || "island".equals(parser.getLocalName())
    || "mountain".equals(parser.getLocalName())
    || "desert".equals(parser.getLocalName())) {
  for (String code :
       parser.getAttributeValue(null, "country").split("␣"))
    if ("USA".equals(code)) {
      waitforUSAlocated = true;
      String id = parser.getAttributeValue(null, "id");
      if (id != null) geoID = id;
      else geoID = geoID + parser.getLocalName();
    }                      // e.g. "river-RheinSource"
}
else if ("located".equals(parser.getLocalName())
         && waitforUSAlocated) {
  if ("USA".equals(parser.getAttributeValue(null, "country"))) {
    waitforUSAlocated = false;
    for (String locprovID :
    (parser.getAttributeValue(null, "province")).split("␣"))
    if (calID.equals(locprovID)) {
      if (parser.getAttributeValue(null, "province")
          .split("␣").length == 1)
      calonlylocateds.add(geoID);
      else calusalocateds.add(geoID);
```

```
        }
      }
    }
    // if getting here (no-break cases and else case): copy
    if ((beforeusa && inneighborcountry == null) || incalifornia) {
      copyStartElement(parser,writer);
    }
    break;
  case XMLStreamConstants.CHARACTERS:
    if ((beforeusa && inneighborcountry == null)|| incalifornia)
    writer.writeCharacters(parser.getText());
    break;
  case XMLStreamConstants.END_ELEMENT:
    if (incalifornia && "unemployment".equals(parser.getLocalName())) {
      // add independence, government and encompassed behind unemployment
      writer.writeEndElement(); writer.writeCharacters("\n");
      writer.writeStartElement("indep_date");
      writer.writeAttribute("from", "USA");
      writer.writeCharacters(getTodayDate());
      writer.writeEndElement(); writer.writeCharacters("\n");
      writer.writeStartElement("government");
      writer.writeCharacters("democracy");
      writer.writeEndElement(); writer.writeCharacters("\n");
      writer.writeStartElement("encompassed");
      writer.writeAttribute("continent", "america");
      writer.writeAttribute("percentage", "100");
      writer.writeEndElement(); // no \n here
    }
    else if   // while parsing caldata
        (incalifornia && "country".equals(parser.getLocalName())) {
      parser.close(); // the caldata parser
      // go back to mparser.
      // It has previously read the first CAL <city ...> StartElement
      parser = mparser;
      incalcity = true;
      writer.writeStartElement("city");
      String cityid = parser.getAttributeValue(null,"id");
      writer.writeAttribute("id",cityid);
      writer.writeAttribute("country","CA");
      if (cityid.equals(capital))
      writer.writeAttribute("is_country_cap","yes");
      writer.writeCharacters("\n");
    }
    else if (incalcity && "city".equals(parser.getLocalName())) {
      incalcity = false;
      writer.writeEndElement(); writer.writeCharacters("\n");
    }
    else if (incalifornia && "province".equals(parser.getLocalName())) {
      incalifornia = false;
      writer.writeEndElement(); writer.writeCharacters("\n");
      // end CAL's country element
    }
    else if (inusprovince && "province".equals(parser.getLocalName())) {
      inusprovince = false;
    }
    else if (inneighborcountry != null
             && "country".equals(parser.getLocalName())) {
      inneighborcountry = null;
    }
    else if (inusa && "country".equals(parser.getLocalName())) {
      inusa = false;
    }
    // in all other cases: just close Element
    else if ((beforeusa && inneighborcountry == null) || incalifornia) {
      writer.writeEndElement();
```

```
        if (parser != caldataparser) writer.writeCharacters("\n");
    }
  }
}

// start second pass:
XMLStreamReader parser =
inputFactory.createXMLStreamReader(new FileInputStream(inputfile));
goOn = true;
while (goOn) {
  int event = parser.next();
  switch(event) {
    case XMLStreamConstants.END_DOCUMENT:
    parser.close();
    writer.flush();
    writer.close();
    goOn = false;
    break;
    case XMLStreamConstants.START_ELEMENT:
    if ("country".equals(parser.getLocalName())) {
      String code = parser.getAttributeValue(null, "car_code");
      if ("USA".equals(code)) {
        inusa = true;
        writer.writeStartElement("country");
        writer.writeAttribute("car_code",
        parser.getAttributeValue(null, "car_code"));
        Integer newarea =
        new Integer(parser.getAttributeValue(null, "area"))
        - new Integer(calAreaS);
        writer.writeAttribute("area", newarea.toString());
        writer.writeAttribute("capital",
        parser.getAttributeValue(null, "capital"));
        writer.writeAttribute("memberships",
        parser.getAttributeValue(null, "memberships"));
        writer.writeCharacters("\n");
      }
      else if (!neighborcodes.isEmpty()) {
        for (int i=0; i< neighborcodes.size(); i++)
        if (((Attribute)
        (neighborcodes.get(i))).getValue().equals(code))
        inneighborcountry = code;
      }
      if (inneighborcountry != null || afterusa) {
        copyStartElement(parser,writer);
      }
      break;
    }
    // end inusa country data
    if (inusa && "province".equals(parser.getLocalName())) {
      provID = parser.getAttributeValue(null, "id");
      capital = parser.getAttributeValue(null, "capital");
      inusprovince = true;
      waitforprovname = true;
      break;
    }
    if (inusprovince && waitforprovname
        && "name".equals(parser.getLocalName())) {
      waitforprovname = false;
      String provname = parser.getElementText();
      if ("California".equals(provname)) {
        incalifornia = true;
      }
      else { // keep the province
        writer.writeStartElement("province");
        writer.writeAttribute("id",provID);
```

```
      writer.writeAttribute("country","USA");
      writer.writeAttribute("capital",capital);
      writer.writeCharacters("\n");
      writer.writeStartElement("name");
      writer.writeCharacters(provname);
      writer.writeEndElement(); writer.writeCharacters("\n");
    }
    break;
  }
  if (inneighborcountry != null
      && "border".equals(parser.getLocalName())) {
    String tocountry = parser.getAttributeValue(null,"country");
    if ("USA".equals(tocountry)) { // border MEX to USA
      writer.writeStartElement("border");
      Float oldlength =
      new Float(parser.getAttributeValue(null,"length"));
      Map<String,Object> vars = new HashMap<String,Object>();
      vars.put("neighbor",null);
      xpath = xpf.compile("./country/border[@country=$neighbor]/@length",
                          Filters.attribute(), vars, (Namespace[]) null);
      xpath.setVariable("neighbor", inneighborcountry);
      Attribute callengthAttr =
      (Attribute) xpath.evaluateFirst(caldata);
      Float callength = new Float(callengthAttr.getValue());
      writer.writeAttribute("country","USA");
      writer.writeAttribute("length", Float.toString(oldlength-callength));
      writer.writeEndElement();writer.writeCharacters("\n");
      // add border to CA:
      writer.writeEmptyElement("border");
      writer.writeAttribute("country","CA");
      writer.writeAttribute("length", callength.toString());
      parser.next(); // ignore this EndElem because <border/> is an empty el.
      writer.writeCharacters("\n");
    }
    else copyStartElement(parser,writer);
    break;
  }
  // US country data: update wrt. CAL data (uses calpop)
  if (inusa && !inusprovince) {
    if ("name".equals(parser.getLocalName())) {
      copyStartElement(parser,writer);
    }
    else if ("population".equals(parser.getLocalName())) {
      String year = parser.getAttributeValue(null, "year");
      parser.next();  // text content
      uspop = new Integer(parser.getText());
      // note: last one stays as uspop
      parser.next();  // endElement
      boolean found = false;
      for (int i=0; i < calpops.size(); i++) {
        if (calpopyears.get(i).equals(year)) {
          found = true;
          calpop = new Integer(calpops.get(i));
          writer.writeStartElement("population");
          writer.writeAttribute("year",year);
          writer.writeCharacters(
          new Integer(uspop - calpop).toString());
          writer.writeEndElement(); writer.writeCharacters("\n");
        }
      }
      if (!found) parser.next(); // read the linebreak for omitted <pop>
      break;
    }
    else if ("government".equals(parser.getLocalName()) ||
    "indep_date".equals(parser.getLocalName())) {
```

```
  copyStartElement(parser,writer);
  parser.next();
  writer.writeCharacters(parser.getText());
  parser.next(); // end element
  writer.writeEndElement(); writer.writeCharacters("\n");
  break;
}
else if ("encompassed".equals(parser.getLocalName()))  {
  copyStartElement(parser,writer);
  parser.next(); // end element
  writer.writeEndElement(); writer.writeCharacters("\n");
  break;
}
else if ( inusa && !inusprovince
&& parser.getAttributeCount() == 0) {
  /* population_growth?,infant_mortality?,
     gdp_total?,gdp_agri?,gdp_ind?,gdp_serv?,inflation?,unemployment?*/
  String elementname = parser.getLocalName();
  writer.writeStartElement(elementname);
  parser.next();
  Float usvalue = new Float(parser.getText());
  xpath = xpf.compile("./country/" + elementname + "/text()");
  Float calvalue =
    new Float(((Text)(xpath.evaluateFirst(caldata))).getTextTrim());
  if ("population_growth".equals(elementname)
      || "infant_mortality".equals(elementname)
      || "unemployment".equals(elementname)) {
    writer.writeCharacters(
       new Float((usvalue * uspop - calvalue * calpop)
                 / (uspop - calpop)).toString());
  }
  else if ("gdp_total".equals(elementname)) {
    usGDP = usvalue;
    writer.writeCharacters(new Float(usvalue - calvalue).toString());
  }
  else if ("gdp_agri".equals(elementname)
      || "gdp_ind".equals(elementname)
      || "gdp_serv".equals(elementname)
      || "inflation".equals(elementname))  {
    writer.writeCharacters(
       new Float((usvalue * usGDP - calvalue * calGDP)
                 / (usGDP - calGDP)).toString());
  }
  parser.next(); // end element
  writer.writeEndElement(); writer.writeCharacters("\n");
  break;
}
else if ("ethnicgroup".equals(parser.getLocalName())
    || "religion".equals(parser.getLocalName())
    || "language".equals(parser.getLocalName()))  {
  String elementname = parser.getLocalName();
  Float oldperc =
        new Float(parser.getAttributeValue(null,"percentage"));
  parser.next();
  String name = parser.getText();
  xpath = xpf.compile("./country/"
               + elementname + "[.='" + name + "']/@percentage");
  Attribute attr = (Attribute)(xpath.evaluateFirst(caldata));
  Double calperc = 0.0;
  if (attr != null) calperc = new Double(attr.getValue());
  writer.writeStartElement(elementname);
  writer.writeAttribute("percentage",
         new Double((oldperc * uspop - calperc * calpop)
  / (uspop - calpop)).toString());
  writer.writeCharacters(name);
```

```
      writer.writeEndElement(); writer.writeCharacters("\n");
      parser.next(); // end element
      break;
    }
    else if ("border".equals(parser.getLocalName())) {
      String country = parser.getAttributeValue(null,"country");
      Double length =  new Double(parser.getAttributeValue(null,"length"));
      xpath = xpf.compile(
              "./country/border[@country␣=␣'" + country + "']/@length");
      Attribute attr = (Attribute)(xpath.evaluateFirst(caldata));
      Double callength = 0.0;
      if (attr != null) callength = new Double(attr.getValue());
      writer.writeEmptyElement("border");
      writer.writeAttribute("country", country);
      writer.writeAttribute("length",
                            new Float(length - callength).toString());
      writer.writeCharacters("\n");
      parser.next(); // ignore this EndElem because <border/> is an empty element
      // after this, add border USA to CAL
      if (!usacalborderadded) {
        usacalborderadded = true;
        xpath = xpf.compile("./country/border[@country␣=␣'USA']/@length");
        String usacallength =
               ((Attribute)(xpath.evaluateFirst(caldata))).getValue();
        writer.writeEmptyElement("border");
        writer.writeAttribute("country", "CA");
        writer.writeAttribute("length", usacallength);
        // writer.writeEndElement(); // no EndElem for EmptyElem <border/>
        writer.writeCharacters("\n");
      }
    }
    break;
  }
  // ######## and now to the organizations: add CAL  where USA and not G-5/G-7
  if ("organization".equals(parser.getLocalName())) {
    inorganization = true;
    copyStartElement(parser,writer);writer.writeCharacters("\n");
    break;
  }
  if (inorganization && "name".equals(parser.getLocalName())) {
    copyStartElement(parser,writer);
    parser.next();
    orgname = parser.getText();
    writer.writeCharacters(orgname);
    break;
  }
  if (inorganization && "members".equals(parser.getLocalName())) {
    String type = parser.getAttributeValue(null, "type");
    String countries = parser.getAttributeValue(null, "country");
    if (!orgname.equals("Group␣of␣5") && !orgname.equals("Group␣of␣7")
    && countries.contains("USA")) { // add CA to the members
      writer.writeStartElement("members");
      writer.writeAttribute("type","type");
      writer.writeAttribute("country", countries.replace("USA","USA␣CA"));
    }
    else
    copyStartElement(parser,writer);
    break;
  }
  // ######## geo things: use calonlylocateds/calusalocateds sets
  if ("river".equals(parser.getLocalName())
      || "lake".equals(parser.getLocalName())
      || "sea".equals(parser.getLocalName())
      || "source".equals(parser.getLocalName())
      || "estuary".equals(parser.getLocalName())
```

```
      || "island".equals(parser.getLocalName())
      || "mountain".equals(parser.getLocalName())
      || "desert".equals(parser.getLocalName())) {
    String id = parser.getAttributeValue(null, "id");
    if (id != null) geoID = id;
    else geoID = geoID + parser.getLocalName();
    // e.g. river-RheinSource
    String countries = parser.getAttributeValue(null, "country");
    geoOnlyCAL = false;
    geoCALUSA = false;
    if (search(calonlylocateds, geoID)) {
      countries = countries.replace("USA","CA");
      geoOnlyCAL = true;
    }
    else if (search(calusalocateds, geoID)) {
      countries = countries.replace("USA","USA␣CA");
      geoCALUSA = true;
    }
    if (geoOnlyCAL || geoCALUSA) {
      waitforUSAlocated = true;
      writer.writeStartElement(parser.getLocalName());
      int n = parser.getAttributeCount();
      for (int i = 0; i < parser.getAttributeCount(); i++) {
        if (!"country".equals(parser.getAttributeLocalName(i)))
        writer.writeAttribute(parser.getAttributeLocalName(i),
        parser.getAttributeValue(i));
        else writer.writeAttribute("country", countries);
      }
    }
    else
      copyStartElement(parser,writer);
    writer.writeCharacters("\n");
    break;
  }
  if ("located".equals(parser.getLocalName())
      && waitforUSAlocated) {
    if ("USA".equals(parser.getAttributeValue(null, "country"))) {
      waitforUSAlocated = false;
      if (geoOnlyCAL) {  // remove the <located>
        // EndElement and "\n" of the <located>
      }
      else { // CALUSA
        writer.writeStartElement("located");
        writer.writeAttribute("country","USA");
        String newprovs = "";
        for (String locprovID :
             parser.getAttributeValue(null, "province").split("␣")) {
          if (!calID.equals(locprovID))
          newprovs = newprovs + locprovID + "␣";
        }
        newprovs = newprovs.substring(0,newprovs.length()-1);
        writer.writeAttribute("province",newprovs);
        writer.writeEndElement(); writer.writeCharacters("\n");
      }
      // note: CAL has no province, thus no located needed for it
      parser.next();  // the end-element of the located
      parser.next();  // the "\n"
    }
    else // it's not the USA-located Element
    copyStartElement(parser,writer);
    break;
  }
  // all others: copy
  if (inneighborcountry != null
      || (inusa && !incalifornia) || afterusa)
```

```
            copyStartElement(parser,writer);
          break;
          case XMLStreamConstants.CHARACTERS:
          if (inneighborcountry != null
              || (inusa && !incalifornia) || afterusa)
            writer.writeCharacters(parser.getText());
          break;
          case XMLStreamConstants.END_ELEMENT:
          if (incalifornia && "province".equals(parser.getLocalName())) {
            incalifornia = false;
            break;
          }
          else if (inusprovince && "province".equals(parser.getLocalName())) {
            inusprovince = false;
            writer.writeEndElement(); writer.writeCharacters("\n");
            break;
          }
          else if ((inneighborcountry != null || inusa || afterusa)
          && "country".equals(parser.getLocalName())) {
            inneighborcountry = null;
            if (inusa) {
              inusa = false;
              afterusa = true;
            }
            writer.writeEndElement(); writer.writeCharacters("\n");
            break;
          }
          if (inneighborcountry != null || (inusa && !incalifornia) || afterusa) {
            writer.writeEndElement(); writer.writeCharacters("\n");
          }
        }
      }
    } catch (Exception e) { e.printStackTrace(); }
  }

  private void copyStartElement(XMLStreamReader parser, XMLStreamWriter writer)
  throws XMLStreamException {
    writer.writeStartElement(parser.getLocalName());
    if (parser.getAttributeCount() > 0) {
      int n = parser.getAttributeCount();
      for (int i = 0; i < parser.getAttributeCount(); i++)
      writer.writeAttribute(parser.getAttributeLocalName(i),
                            parser.getAttributeValue(i));
    }
  }

  private boolean search(List<String> list, String s) {
    // Collection.countains uses ==, but here equals is needed
    boolean found = false;
    for (String x:list)
    if (s.equals(x)) found = true;
    return found;
  }

  private String getTodayDate() {
    DateFormat format = new SimpleDateFormat();
    return format.format(new Date());   }

  public static void main(String[] args) {
    CalexitStAXTwoPass calexit =
    new CalexitStAXTwoPass("mondial.xml", System.out);
    calexit.startParsing();
  }
}
```

```
import java.io.InputStreamReader;
import java.io.BufferedReader;

// Derived from CalexitStAXOnePass, using a jdom for intermediate
// storage and updates (mainly, USA)
import java.io.InputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.OutputStream;
import java.io.PipedInputStream;
import java.io.PipedOutputStream;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;
import java.util.LinkedList;
import java.util.Map;
import java.util.HashMap;

import java.io.PipedInputStream;
import java.io.PipedOutputStream;
import java.io.BufferedOutputStream;

import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLOutputFactory;
import javax.xml.stream.XMLStreamConstants;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamReader;
import javax.xml.stream.XMLStreamWriter;

import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.Attribute;
import org.jdom2.Text;
import org.jdom2.Namespace;
import org.jdom2.input.SAXBuilder;
import org.jdom2.xpath.XPathFactory;
import org.jdom2.xpath.XPathExpression;
import org.jdom2.filter.Filters;
import org.jdom2.JDOMException;
import org.jdom2.output.XMLOutputter;

public class CalexitStAXDOM {

  FileInputStream inputStream;
  OutputStream outputStream;
  XMLInputFactory inputFactory = XMLInputFactory.newInstance();
  Document caldata = null;
  List<Attribute> neighborcodes = new LinkedList<Attribute>();  // empty
  Document usadoc = null;
  Element usa = null;
  Float calGDP = null;
  Float usGDP = null;
  Integer calpop = null;
  Integer uspop = null;
  Float callength = null;
  String usmemberships = null;
  String calAreaS = null;
  Document geodoc = null;
  Element geo = null;
  String geoname = null;
  boolean geoOnlyCAL = false;
  boolean geoOnlyUSA = false;
```

```
boolean geoCALUSA = false;
List<String> calpopyears = new LinkedList<String>();
List<String> calpops = new LinkedList<String>();

XPathFactory xpf = XPathFactory.instance();
XPathExpression xpath = null; // only for multiple short term use

public enum tasks { unchanged, handleCalCountry, handleCalData };

public CalexitStAXDOM(FileInputStream in, OutputStream out) {
  this.inputStream = in;
  this.outputStream = out;
}

public void startParsing() {
  SAXBuilder builder = new SAXBuilder();
  File xmlFile = new File("caldata.xml");
  try {
    caldata = (Document) builder.build(xmlFile);
    xpath = xpf.compile("./country/border/@country");
    neighborcodes = xpath.evaluate(caldata); // includes "USA"
    xpath = xpf.compile("./country/gdp_total/text()");
    calGDP = new Float(((Text)(xpath.evaluateFirst(caldata))).getTextTrim());

    XMLOutputFactory outputFactory = XMLOutputFactory.newInstance();
    XMLStreamReader mparser = inputFactory.createXMLStreamReader(inputStream);
    XMLStreamWriter mwriter = outputFactory.createXMLStreamWriter(outputStream);
    XMLStreamReader caldataparser =
      inputFactory.createXMLStreamReader(new FileInputStream("caldata.xml"));
    boolean goOn = true;
    boolean inusa = false;
    boolean inusprovince = false;
    boolean waitforprovname = false;
    boolean incalifornia = false;
    boolean incalcity = false;
    boolean calfirstcityover = false;
    String calID = null;
    String inneighborcountry = null;
    boolean inorganization = false;
    String orgname = null;
    boolean waitforUSAlocated = false;
    String provID = "";
    String capital = "";

    // write US Geo things intermediately in a Buffered stream while
    // waiting for "located"   (also used for US country data before)
    PipedOutputStream geoPOS = null;
    BufferedOutputStream geoBOS = null;
    PipedInputStream geoPIS = null;
    PipedOutputStream usaPOS = null;
    BufferedOutputStream usaBOS = null;
    PipedInputStream usaPIS = null;
    XMLStreamWriter usawriter = null;

    // The following is first written into temporary files/streams/DOM:
    // USA country data (to be updated later when CAL population is known
    // for weighing infant mortality, religions/languages/ethnicgroups)
    // and its provinces occurring before CAL.
    // California is directly written as a new country to mondial,
    // then the buffered USA data is adapted/flushed an the rest is
    // directly written to mondial.

    XMLStreamReader parser = mparser;
    XMLStreamWriter writer = mwriter;
    writer.writeStartDocument("UTF-8", "1.0");
```

```
writer.writeCharacters("\n");

while (goOn) {
  int event = parser.next();
  switch(event) {
  case XMLStreamConstants.DTD:
    String doctypedecl = parser.getText();
    writer.writeDTD(doctypedecl);
    writer.writeCharacters("\n");
    break;
  case XMLStreamConstants.END_DOCUMENT:
    parser.close();
    writer.flush();
    writer.close();
    goOn = false;
    break;
  case XMLStreamConstants.START_ELEMENT:
    if ("country".equals(parser.getLocalName())) {
      String code = parser.getAttributeValue(null, "car_code");
      if ("USA".equals(code)) {
        inusa = true;
        usmemberships = parser.getAttributeValue(null, "memberships");
        usaPIS = new PipedInputStream(); // <- this will be read, now by the DOM
        usaPOS = new PipedOutputStream(usaPIS);
        usaBOS = new BufferedOutputStream(usaPOS,100000); // write tmp BOS->POS->PIS
        usawriter = outputFactory.createXMLStreamWriter(usaBOS);
        writer = usawriter;
      }
      else if (!neighborcodes.isEmpty()) {
        for (int i=0; i< neighborcodes.size(); i++)
          if (((Attribute)(neighborcodes.get(i))).getValue().equals(code))
            inneighborcountry = code;
      }
      copyStartElement(parser,writer); writer.writeCharacters("\n");
      break;
    }
    else if (inusa && !inusprovince // all elements between <pop> and first <province>
             && !"population".equals(parser.getLocalName())
             && !"province".equals(parser.getLocalName())) {
      copyStartElement(parser,writer); // writer is geowriter;
          // change all of them later
      break;
    }
    else if (inusa && "province".equals(parser.getLocalName())) {
      provID = parser.getAttributeValue(null, "id");
      capital = parser.getAttributeValue(null, "capital");
      inusprovince = true;
      waitforprovname = true;
      break;
    }
    else if (inusprovince && waitforprovname
             && "name".equals(parser.getLocalName())) {
      waitforprovname = false;
      String provname = parser.getElementText();
      if ("California".equals(provname)) {
        incalifornia = true;
        writer.writeEndElement();  // </country> for USA to usawriter->usaPOS
        writer.writeEndDocument();  // end USA doc
        writer = mwriter;
        calID = provID;
        writer.writeStartElement("country");
        writer.writeAttribute("car_code","CA");
        writer.writeAttribute("capital",capital);
        writer.writeAttribute("memberships",  // dirty. But I don't want to wait for the orgs.
                usmemberships.replace("␣org-G-5","").replace("␣org-G-7",""));
```

```
    }
    else {
      writer.writeStartElement("province");
      writer.writeAttribute("id",provID);
      writer.writeAttribute("country","USA");
      writer.writeAttribute("capital",capital);
      writer.writeCharacters("\n");
      writer.writeStartElement("name");
      writer.writeCharacters(provname);
      writer.writeEndElement();writer.writeCharacters("\n");
    }
    break;
  }
  else if (incalifornia && "area".equals(parser.getLocalName())) {
    calAreaS = parser.getElementText();
    writer.writeAttribute("area",calAreaS);
    writer.writeCharacters("\n");
    writer.writeStartElement("name");
    writer.writeCharacters("California");
    writer.writeEndElement(); writer.writeCharacters("\n");
    break;
  }
  else if (incalifornia && !incalcity
           && "population".equals(parser.getLocalName())) {
    copyStartElement(parser,writer);
    String year = parser.getAttributeValue(null, "year");
    calpopyears.add(year);
    parser.next();  // text content
    String calpopS = parser.getText();
    writer.writeCharacters(calpopS);
    calpops.add(calpopS);
    calpop = new Integer(calpopS); // note: last one stays as calpop
    parser.next(); // </population>
    writer.writeEndElement(); writer.writeCharacters("\n");
    break;
  }
  else if (incalifornia && "city".equals(parser.getLocalName())) {
    if (!calfirstcityover) {
      calfirstcityover = true;
      copyFromXMLStream(caldataparser, writer, tasks.handleCalData, "");
    }
    incalcity = true;
    writer.writeStartElement("city");
    String cityid = parser.getAttributeValue(null,"id");
    writer.writeAttribute("id",cityid);
    writer.writeAttribute("country","CA");
    if (cityid.equals(capital))
        writer.writeAttribute("is_country_cap","yes");
    writer.writeCharacters("\n");
    break;
  }
  else if (inneighborcountry != null
           && "border".equals(parser.getLocalName())) {
    String tocountry = parser.getAttributeValue(null,"country");
    if ("USA".equals(tocountry)) { // border MEX to USA
      writer.writeStartElement("border");
      Float oldlength =
          new Float(parser.getAttributeValue(null,"length"));
      Map<String,Object> vars = new HashMap<String,Object>();
      vars.put("neighbor",null);
      xpath = xpf.compile("./country/border[@country=$neighbor]/@length",
                          Filters.attribute(), vars, (Namespace[]) null);
      xpath.setVariable("neighbor", inneighborcountry);
      Attribute callengthAttr =
          (Attribute) xpath.evaluateFirst(caldata);
```

```
      callength = new Float(callengthAttr.getValue());
      writer.writeAttribute("country","USA");
      writer.writeAttribute("length",
                            Float.toString(oldlength-callength));
      writer.writeEndElement();writer.writeCharacters("\n");
      // add border to CA:
      writer.writeEmptyElement("border");
      writer.writeAttribute("country","CA");
      writer.writeAttribute("length", callength.toString());
      parser.next(); // ignore this EndElem because <border/> is an empty el.
      writer.writeCharacters("\n");
    }
    else copyStartElement(parser,writer);
    break;
  }
  // ######## and now to the organizations:
  else if ("organization".equals(parser.getLocalName())) {
    inorganization = true;
    copyStartElement(parser,writer);writer.writeCharacters("\n");
    break;
  }
  else if (inorganization && "name".equals(parser.getLocalName())) {
    copyStartElement(parser,writer);
    parser.next();
    orgname = parser.getText();
    writer.writeCharacters(orgname);
    break;
  }
  else if (inorganization && "members".equals(parser.getLocalName())) {
    String type = parser.getAttributeValue(null, "type");
    String countries = parser.getAttributeValue(null, "country");
    if (!orgname.equals("Group␣of␣5") && !orgname.equals("Group␣of␣7")
        && countries.contains("USA")) { // add CA to the members
        writer.writeStartElement("members");
        writer.writeAttribute("type","type");
        writer.writeAttribute("country",
                              countries.replace("USA","USA␣CA"));
    }
    else
        copyStartElement(parser,writer);
    break;
  }
  // ######## and now to the geo things:
  // disadvantage of StAX against XSLT: all geos must explicitly be listed
  // even worse disadvantage: read down to "located" to know whether
  //       located in CAL
  // => use again the tmp BufferedStream
  else if ("river".equals(parser.getLocalName())
       || "lake".equals(parser.getLocalName())
       || "sea".equals(parser.getLocalName())
       || "source".equals(parser.getLocalName())
       || "estuary".equals(parser.getLocalName())
       || "island".equals(parser.getLocalName())
       || "mountain".equals(parser.getLocalName())
       || "desert".equals(parser.getLocalName())) {
    geoname = parser.getLocalName();
    for (String code : parser.getAttributeValue(null, "country").split("␣"))
      if ("USA".equals(code)) {
        waitforUSAlocated = true;
        geoPIS = new PipedInputStream(); // <- this will be read, now by the DOM
        geoPOS = new PipedOutputStream(geoPIS);
        geoBOS = new BufferedOutputStream(geoPOS,100000); // write tmp BOS->POS->PIS
        XMLStreamWriter geowriter = outputFactory.createXMLStreamWriter(geoBOS);
        writer = geowriter;
      }
```

```
      copyStartElement(parser,writer); writer.writeCharacters("\n");
      break;
    }
    else if ("located".equals(parser.getLocalName())
             && waitforUSAlocated) {
      if ("USA".equals(parser.getAttributeValue(null, "country"))) {
        waitforUSAlocated = false;
        geoOnlyUSA = true;
        geoOnlyCAL = false;
        geoCALUSA = false;
        for (String locprovID :
             parser.getAttributeValue(null, "province").split("␣"))
          if (calID.equals(locprovID)) {
            if (parser.getAttributeValue(null, "province")
                  .split("␣").length == 1)
              { geoOnlyCAL = true; geoOnlyUSA = false; }
            else  { geoCALUSA = true; geoOnlyUSA = false; }
          }

        // Now, we know whether the geo is in US/CAL
        if (geoOnlyUSA) { // <located USA/> unchanged
           copyStartElement(parser,writer);
           writer.writeEndElement();
           writer.writeCharacters("\n");
        }
        else if (geoCALUSA) {  // prov-california rausnehmen
          writer.writeEmptyElement("located");
          writer.writeAttribute("country","USA");
          String newprovs = "";
          for (String locprovID :
               parser.getAttributeValue(null, "province").split("␣")) {
            if (!calID.equals(locprovID))
              newprovs = newprovs + locprovID + "␣";
          }
          newprovs = newprovs.substring(0,newprovs.length()-1);
          writer.writeAttribute("province",newprovs);
          writer.writeCharacters("\n");
        }  // note: CAL has no province, thus no located needed for it
        writer.writeEndElement(); // </river/lake/sea>
        writer.writeEndDocument();
        parser.next(); // read the EndElement event of the <located/>
        writer = mwriter; // write the following again to mondial

        final BufferedOutputStream geoBOS2 = geoBOS;
        new Thread() {
          public void run(){
            try{ geoBOS2.flush();
                 geoBOS2.close();
            } catch (Exception e) { e.printStackTrace(); }}}.start();
        // note: SAX Builder continues reading after end of Document ...
        //   ... if geoBOS.close() is not done above, it raises "write end dead";
        // => BOS cannot be reused; reinitialize each time
        geodoc = (Document) builder.build(geoPIS);
        geo = geodoc.getRootElement();  // <river/lake/sea/...>
        String countries = geo.getAttributeValue("country");
        if (geoOnlyCAL)
            countries = countries.replace("USA","CA");
        else if (geoCALUSA)
            countries = countries.replace("USA","USA␣CA");
        geo.setAttribute("country", countries);

        geoPIS = new PipedInputStream(); // <- this will be read by the other StAX
        geoPOS = new PipedOutputStream(geoPIS);
        // geoBOS = new BufferedOutputStream(System.out, 100000); // check -> output
        geoBOS = new BufferedOutputStream(geoPOS, 1000000); // BOS -> POS -> PIS
```

```
    // outputter.output(geodoc.getRootElement(), System.out);
    final BufferedOutputStream geoBOS3 = geoBOS;
    new Thread() {
      public void run(){
          try{
            XMLOutputter outputter = new XMLOutputter();
            outputter.output(geodoc.getRootElement(), geoBOS3);
            geoBOS3.flush();  // revised Geo DOM into Stream
            geoBOS3.close();
          } catch (Exception e) { e.printStackTrace(); }}}.start();
    // note: the final </country> must be omitted -> run through another StAX round
    XMLStreamReader geoparser = inputFactory.createXMLStreamReader(geoPIS);
    copyFromXMLStream(geoparser, writer, tasks.unchanged, geoname);
  }
  else // not the USA-located Element
      copyStartElement(parser,writer);
  break;
}
// all others: copy
else copyStartElement(parser,writer);
  break;
case XMLStreamConstants.CHARACTERS:
  writer.writeCharacters(parser.getText());
  break;
case XMLStreamConstants.END_ELEMENT:
  if (incalcity && "city".equals(parser.getLocalName())) {
    incalcity = false;
    writer.writeEndElement(); writer.writeCharacters("\n");
  break;
  }
  else if (incalifornia && "province".equals(parser.getLocalName())) {
    incalifornia = false;
    // end CAL's country element
    writer.writeEndElement(); writer.writeCharacters("\n");
    writer = mwriter; // write the following again to mondial

    // Get USA country data from usa buffered Stream:
    //   opening <country ... > tag (update area) + <name>...</>
    final BufferedOutputStream usaBOS2 = usaBOS;
    new Thread() {
      public void run(){
          try{ usaBOS2.flush();  // -> usaPOS -> usaPIS
              usaBOS2.close();
          } catch (Exception e) { e.printStackTrace(); }}}.start();
    // builder already exists
    usadoc = (Document) builder.build(usaPIS);
    usa = usadoc.getRootElement();  // <country>
    usa.setAttribute("area",
                     Float.toString(Float.parseFloat(usa.getAttributeValue("area")) -
                     Float.parseFloat(calAreaS)));
    List<Element> pops = new LinkedList<Element>(usa.getChildren("population"));
    for (Element popEl:pops) {
        String year = popEl.getAttributeValue("year");
        boolean found = false;
        for (int i=0; i < calpops.size(); i++) {
            if (calpopyears.get(i).equals(year)) {
                found = true;
                calpop = new Integer(calpops.get(i));
            }
        }
        if (found) {
            uspop = Integer.parseInt(popEl.getTextTrim());
            popEl.setText(Integer.toString(uspop - calpop));
        }
        else popEl.detach();
```

```
    }
    List<Element> borders = new LinkedList<Element>(usa.getChildren("border"));
    for (Element borderEl:borders) {
        String country = borderEl.getAttributeValue("country");
        Double length = new Double(borderEl.getAttributeValue("length"));
        xpath = xpf.compile("./country/border[@country = '" + country + "']/@length");
        Attribute attr = (Attribute)(xpath.evaluateFirst(caldata));
        Double callength = 0.0;
        if (attr != null) callength = new Double(attr.getValue());
        borderEl.setAttribute("length", new Float(length - callength).toString());
    }
    // DEBUG
    // XMLOutputter outputter1 = new XMLOutputter();
    // outputter1.output(usa, System.out);
    weighByPop("population_growth");
    weighByPop("infant_mortality");
    weighByPop("unemployment");

    // GDPs
    Element el = usa.getChild("gdp_total");
    usGDP = Float.parseFloat(el.getTextTrim());
    el.setText(Float.toString(usGDP - calGDP));
    weighByGDP("gdp_agri");
    weighByGDP("gdp_ind");
    weighByGDP("gdp_serv");
    weighByGDP("inflation");

    handleByGroups("ethnicgroup");
    handleByGroups("religion");
    handleByGroups("language");

    int index = usa.indexOf(usa.getChild("border"));
    usa.addContent(index,
                   new Element("border").setAttribute("country","CA")
                   .setAttribute("length",callength.toString()));

    // output SPAETER IN DEN WRITER REINPFEIFEN, END COUNTRY DABEI LOESCHEN

    // ... and write out usaBOS with cities and provinces (everything that
    //   occurred in mondial.xml before California) to writer
    //   note: the final </country> must be omitted -> through another StAX round
    usaPIS = new PipedInputStream(); // <- this will be read, now by the other StAX
    usaPOS = new PipedOutputStream(usaPIS);
    // usaBOS = new BufferedOutputStream(System.out, 100000); // check -> output
    usaBOS = new BufferedOutputStream(usaPOS, 1000000); // BOS -> POS -> PIS
    // outputter.output(usadoc.getRootElement(), System.out);
    final BufferedOutputStream usaBOS3 = usaBOS;
    new Thread() {
      public void run(){
          try{
            XMLOutputter outputter = new XMLOutputter();
            outputter.output(usadoc.getRootElement(), usaBOS3);
            usaBOS3.flush();  // revised USA DOM into Stream
            usaBOS3.close();
          } catch (Exception e) { e.printStackTrace(); }}}.start();
    // note: the final </country> must be omitted -> through another StAX round
    XMLStreamReader usaparser =
        inputFactory.createXMLStreamReader(usaPIS);
    copyFromXMLStream(usaparser, writer, tasks.unchanged, "country");
    break;
  }
  else if (inusprovince && "province".equals(parser.getLocalName())) {
    inusprovince = false;
    writer.writeEndElement(); writer.writeCharacters("\n");
    break;
```

```
            }
            else if ("organization".equals(parser.getLocalName())) {
              inorganization = false;
              writer.writeEndElement(); writer.writeCharacters("\n");
              break;
            }
            else if (inusa && "country".equals(parser.getLocalName())) {
              inusa = false;
                // no "break" ...
            }
            else if ("country".equals(parser.getLocalName())) {
              inneighborcountry = null;
              // no "break" ...
            }
            // in all cases that did not have a "break": close Element
            writer.writeEndElement();
            writer.writeCharacters("\n");
            break;
          }
      }
    } catch (Exception e) { e.printStackTrace(); }
  }

  private void copyStartElement(XMLStreamReader parser, XMLStreamWriter writer)
      throws XMLStreamException {
    writer.writeStartElement(parser.getLocalName());
    if (parser.getAttributeCount() > 0) {
      int n = parser.getAttributeCount();
      for (int i = 0; i < parser.getAttributeCount(); i++)
        writer.writeAttribute(parser.getAttributeLocalName(i),
                              parser.getAttributeValue(i));
    }
  }

  private void copyFromXMLStream(XMLStreamReader parser,
                                 XMLStreamWriter writer, tasks task, String endname)
      throws Exception {
    boolean goOn = true;
    // System.out.println("copyFromXMLStream");
    while (goOn) {
      int event = parser.next();
      switch (event) {
      case XMLStreamConstants.START_ELEMENT:
        if (task == tasks.handleCalData
            && "country".equals(parser.getLocalName())) { // in caldata.xml
          parser.next(); // ignore a "/n"
          break; // ignore;
        }
        copyStartElement(parser, writer);
        break;
      case XMLStreamConstants.CHARACTERS:
        writer.writeCharacters(parser.getText());
        break;
      case XMLStreamConstants.END_ELEMENT:
        if (task == tasks.handleCalData
                 && "unemployment".equals(parser.getLocalName())) {
          // add independence, government and encompassed behind unemployment
          writer.writeEndElement(); writer.writeCharacters("\n");
          writer.writeStartElement("indep_date");
          writer.writeAttribute("from", "USA");
          writer.writeCharacters(getTodayDate());
          writer.writeEndElement(); writer.writeCharacters("\n");
          writer.writeStartElement("government");
          writer.writeCharacters("democracy");
          writer.writeEndElement(); writer.writeCharacters("\n");
```

```
          writer.writeStartElement("encompassed");
          writer.writeAttribute("continent", "america");
          writer.writeAttribute("percentage", "100");
          writer.writeEndElement(); // no \n here necessary
          break;
        }
        else if (task == tasks.handleCalData
                 && "country".equals(parser.getLocalName())) { // in caldata.xml
          goOn = false;
          break;
        }
        else if (task == tasks.unchanged
                 && endname.equals(parser.getLocalName())) { // from USA/GEO DOM
          goOn = false;
          break;
        }
        else if (task == tasks.handleCalCountry
                 && "country".equals(parser.getLocalName()))
          goOn = false;
        writer.writeEndElement();
      }
    }
  }

  // same as in the DOM case
  private void weighByPop(String elementname) {
    Element el = usa.getChild(elementname);
    Float usvalue = Float.parseFloat(el.getTextTrim());
    Float calvalue =
        Float.parseFloat(caldata.getRootElement().getChild(elementname).getTextTrim());
    el.setText(Float.toString(
                (usvalue * uspop - calvalue * calpop) / (uspop - calpop)));
  }
  private void weighByGDP(String elementname) {
    Element el = usa.getChild(elementname);
    Float usvalue = Float.parseFloat(el.getTextTrim());
    Float calvalue =
        Float.parseFloat(caldata.getRootElement().getChild(elementname).getTextTrim());
    el.setText(Float.toString(
                (usvalue * usGDP - calvalue * calGDP) / (usGDP - calGDP)));
  }
  private void handleByGroups(String elementname) throws JDOMException {
    List<Element> uselems = usa.getChildren(elementname);
    for (Element uselem:uselems) {
      String name = uselem.getTextTrim();
      xpath = xpf.compile("./country/" + elementname+"[.='"+name+"']/@percentage");
      Attribute calpercAt = (Attribute)xpath.evaluateFirst(caldata);
      if (calpercAt != null) {
        Float usvalue = Float.parseFloat(uselem.getAttributeValue("percentage"));
        Float calvalue = Float.parseFloat(calpercAt.getValue());
        uselem.setAttribute("percentage",
         Float.toString((usvalue * uspop - calvalue * calpop)
                        / (uspop - calpop)));
      }
    }
  }
  private String getTodayDate() {
    DateFormat format = new SimpleDateFormat();
    return format.format(new Date());   }

  public static void main(String[] args) {
    try {
      CalexitStAXDOM calexit =
          new CalexitStAXDOM(new FileInputStream("mondial.xml"), System.out);
      calexit.startParsing();
```

```
    } catch (Exception e) { e.printStackTrace(); }
 }}
```

4. Unit: XML Processing with Java (II)

Exercise 4.1 (XML Digester) Use the XML Digester in a Java class that creates an internal data structure about seas, rivers and their tributaries (including lakes) and generates an output similar to that in Exercise 2.1.
Additionally, this output should show for every river the sum of the length of all rivers flowing into it (directly or indirectly).

```
import java.io.File;
import java.util.HashMap;
import java.util.TreeSet;
import java.util.Set;
import java.util.HashSet;

import org.apache.commons.digester3.Digester;
import org.apache.commons.digester3.ObjectCreateRule;

/* Strategy for order-tributaries-by-elevation:
 * a TreeSet with the elevations (-> descending iterator)
 * a Map<elev,Set<Rivers>> with the rivers for each elevation
   (several rivers might flow in at the same elevation, e.g. for a lake)
 */

public class DigesterRiverLength {

  public static void renderWater(Water start, String indent) {
  System.out.println(indent + "-" + start.toString() + "␣NetwLength:␣"
                     + start.getTransitiveLength());
  for (float e : start.tributaryElevs) {
    for (Water r: start.tributaries.get(e))
        renderWater(r, "␣"+indent);
    }
  }

  public static class WaterMap extends HashMap<String, Water> {
    public void addWater(Water r) {
      this.put(r.getWaterId(), r);
    }
    public void reverseConnectWaters() {
      for (Water r : values()) {
        for(String dest_id : r.to) {
          Water d = get(dest_id);
          if (d != null)
          d.addTributary(r);
        }
      }
    }
  }
  public static class Water {
    String waterId = null;
    String name = null;
    float length = 0;
    float elevation = 0; // elevation of the estuary
    final TreeSet<Float> tributaryElevs;
    final HashMap<Float,Set<Water>> tributaries;
    final TreeSet<String> to;

    public Water() {
```

```
    this.tributaryElevs = new TreeSet<Float>();
    this.tributaries = new HashMap<Float, Set<Water>>();
    this.to = new TreeSet<String>();
  }
  public void setWaterId(String id) { this.waterId = id; }
  public void setName(String name) { this.name = name; }
  public void setLength(float length) { this.length = length; }
  public void setElevation(float elev) { this.elevation = elev; }
  public String getWaterId() { return waterId; }
  public float getLength() { return length; }
  public float getTransitiveLength() {
    float t_length = length;
    for (Set<Water> rset : tributaries.values())
      for (Water r : rset)
        t_length += r.getTransitiveLength();
    return t_length;
  }
  public void addTributary(Water r) {
    tributaryElevs.add(r.elevation);
    if (tributaries.get(r.elevation) == null)
      tributaries.put(new Float(r.elevation), new HashSet<Water>());
    tributaries.get(r.elevation).add(r);
  }
  public void addDestination(String water) {
    to.add(water);
  }
  @Override
  public boolean equals(Object obj) {
    if (this == obj)
    return true;
    if (obj == null)
    return false;
    if (getClass() != obj.getClass())
    return false;
    Water other = (Water) obj;
    if (waterId == null) {
      if (other.waterId != null)
      return false;
    } else if (!waterId.equals(other.waterId))
    return false;
    return true;
  }
  @Override
  public String toString() {
    return name + "␣(" + length + "km␣at␣" + elevation + "m)";
  }
};

public static void main(String[] args) {

  File res = new File("mondial.xml");
  System.out.println("Parsing:␣"+res);

  Digester digester = new Digester();
  // note: Digester patternd do no support mondial/(river|lake|sea)
  digester.push(new WaterMap());
  digester.addObjectCreate("mondial/sea", Water.class);
  digester.addSetProperties("mondial/sea", "id", "waterId");
  digester.addBeanPropertySetter("mondial/sea/name");
  digester.addSetNext("mondial/sea", "addWater");
  // rivers:
  digester.addObjectCreate("mondial/river", Water.class);
  digester.addSetProperties("mondial/river", "id", "waterId");
  digester.addBeanPropertySetter("mondial/river/name");
  digester.addBeanPropertySetter("mondial/river/length");
```

```
    digester.addBeanPropertySetter("mondial/river/estuary/elevation");
    digester.addCallMethod("mondial/river/to", "addDestination", 1);
    digester.addCallParam("mondial/river/to", 0, "water");
    digester.addSetNext("mondial/river", "addWater");
    // handle lakes like rivers
    digester.addObjectCreate("mondial/lake", Water.class);
    digester.addSetProperties("mondial/lake", "id", "waterId");
    digester.addBeanPropertySetter("mondial/lake/name");
    digester.addBeanPropertySetter("mondial/lake/elevation");
    digester.addCallMethod("mondial/lake/to", "addDestination", 1);
    digester.addCallParam("mondial/lake/to", 0, "water");
    digester.addSetNext("mondial/lake", "addWater");
    try {
      digester.setValidating(false);
      final WaterMap water_map = digester.parse(res);
      water_map.reverseConnectWaters();
      for (String waterid: water_map.keySet()) {
         if (water_map.get(waterid).to.isEmpty())
            renderWater(water_map.get(waterid), "");
      }
      // renderWater(water_map.get("sea-Nordsee"), "");
      // renderWater(water_map.get("river-Rhein"), "");
    } catch (Exception e) { e.printStackTrace(); }
  }
}
// note: after-comma decimals!
// if java datatype is integer, they are translated to 0.
// Amazonas 33676
// Rhein     4515.3  -- note: Bregenzer Ach -> Bodensee -> Rhein
// Donau    14473.9  -- note: Inn and Ilz tributaries at same elev (291m, Passau)
// Nil      12626  -- note: several rivers flowing into the same lakes at same elev
// Zaire    21809
```

Exercise 4.2 (XML/JAXB/DOM/XSLT) Application: Personal schedule

This exercise will be reused for a Web Service.

Consider an XML structure for representing a personal schedule ("Terminkalender") with entries (using xsd:dateTime):

```
<?xml version="1.0" encoding="UTF-8"?>
<schedule name="John Doe">
  <year n="2016">
    <month n="12">
      <day n="7">
        <entry starttime="16:00:00" duration="PT02H">
          <name>XML Lab</name>
          <description>weekly course meeting</description>
          <location>IFI 1.101</location>
        </entry>
      </day>
      <day n="8">
        <entry starttime="11:00:00" duration="PT50M">
          <name>Discussion MSc Thesis X.Y.</name>
        </entry>
      </day>
    </month>
  </year>
</schedule>
```

- Each schedule contains zero or more year elements.

- Each year element contains zero or more month elements.
- Each month element contains zero or more day elements.
- Each day element contains zero or more entry elements.
- Each entry element, describes a certain date, containing information about starting time, duration, location and a textual description of the date.
- Appointments do not span over more than one day.
- The month and day elements appear in their correct temporal order inside the document.

The exercise consists of the following steps:

a) Design an XML Schema (use the dateTime datatype from XML Schema), and validate an example instance.

b) Generate basic classes and interfaces for the schedule application, using the JAXB Schema binding compiler on your XML schema.

c) Unmarshal your example schedule.xml file into memory using JAXB.

d) Write a method insertEntry(date, time, duration, title) that inserts a new appointment into the internal schedule. If the appointment is collision-free, it should be inserted, otherwise the method should return "false".

e) Marshal the in-memory schedule into an in-memory JDOM Tree, again using JAXB.

f) Write an XSLT styhesheet that transforms an XML schedule instance (which can be given as a file or an JDOM instance) into an HTML output presentation of the appointments if the given month (HTML table or list).
Write a method that applies the stylesheet directly to the JDOM tree and outputs the result into a file (use e.g. the javax.xml.transform classes).

Notes:

- The command: "xjc -help" issued on the command line displays usage and possible options.
- At `/afs/informatik.uni-goettingen.de/course/xml-prakt/xml/JAXB-README` you find an explanation for installing, testing and using JAXB.
- The Java WebService Tutorial at http://java.sun.com/webservices/tutorial.html provides some help for working with JAXB.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="schedule">
      <xs:complexType>
         <xs:sequence>
            <xs:element maxOccurs="unbounded" minOccurs="0" ref="year"/>
         </xs:sequence>
         <xs:attribute name="name" type="xs:string"/>
      </xs:complexType>
   </xs:element>

   <xs:element name="year">
      <xs:complexType>
         <xs:sequence>
            <xs:element maxOccurs="unbounded" minOccurs="0" ref="month"/>
         </xs:sequence>
         <xs:attribute name="n" type="xs:integer"/>
      </xs:complexType>
```

```
   </xs:element>

<xs:element name="month">
      <xs:complexType>
         <xs:sequence>
            <xs:element maxOccurs="unbounded" minOccurs="0" ref="day"/>
         </xs:sequence>
         <xs:attribute name="n" type="xs:integer"/>
      </xs:complexType>
   </xs:element>

<xs:element name="day">
      <xs:complexType>
         <xs:sequence>
            <xs:element maxOccurs="unbounded" minOccurs="0" ref="entry"/>
         </xs:sequence>
         <xs:attribute name="n" type="xs:integer"/>
      </xs:complexType>
   </xs:element>

   <xs:element name="entry">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="name"/>
            <xs:element minOccurs="0" ref="description"/>
            <xs:element minOccurs="0" ref="location"/>
         </xs:sequence>
         <xs:attribute name="starttime" type="xs:time"/>
         <xs:attribute name="duration" type="xs:duration"/>
         <xs:attribute name="id" type="xs:ID"/>
      </xs:complexType>
   </xs:element>
<xs:element name="name" type="xs:string"/>
<xs:element name="description" type="xs:string"/>
<xs:element name="location" type="xs:string"/>
</xs:schema>
```

```
package schedulejaxb;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;

public class Entry {
    private String name = "", description = "", location = "", id = "";
    private Time starttime;
    private Time duration;

    public Entry(String name, String id, Time starttime, Time duration) {
        super();
        this.name = name;
        this.id = id;
        this.starttime = starttime;
        this.duration = duration;
    }
```

```
        StreamResult result = new StreamResult(outputFile);
        try {
            Source source = new DOMSource(schedule);
            Transformer transformer = factory.newTransformer(xsltSource);
            transformer.transform(source, result);
        } catch (TransformerConfigurationException e) {
            e.printStackTrace();
        } catch (TransformerException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

Exercise 4.3 (Calexit in JAXB)

- Do the Calexit exercise using JAXB.
  (`mondial.xsd` is available on the Web site)
- Why is it not necessarily a good idea to do it with the Digester?

- ```
  Goto directory Calexit:
  *  copy mondial.dtd, mondial.xml, mondial.xsd there
  >  mkdir gensrc
  > xjc -p JAXBmondial mondial.xsd -d gensrc
    -- omitting -p would create "generated" default
  * creates gensrc/JAXBmondial
  > javac -d . `find gensrc -name '*.java'`
  * creates JAXBmondial/ with the class files.
  > copy CalexitJAXB.java into JAXBmondial/  (must be in package JAXBmondial)
  > javac JAXBmondial/CalexitJAXB.java
  > java JAXBmondial/CalexitJAXB, generates output.xml
  ```
- Comments on CalexitJAXB.java:
  - I used the CalexitJDOM.java code as a base. The proceeding by acting on the graph structure is basically the same.
  - JAXB unmarshalling of mondial.xml without any problems; even IDREFs are references to the respective java objects,
  - caldata.xml cannot be unmarshalled completely, because border/@country cannot to MEX and USA be dereferenced (set to null)
    $\Rightarrow$ requires to put it also into a DOM.
  - finding all geo objects located in CAL requires an explicit loop over all their located elements. Even more, it requires a separate loop for each geo class, because there is no superclass.
  - processing numerical values: JAXB assigns BigDecimal and BigInteger types, whose handling for arithmetics is tedious and requires some explicit conversions.
  - handling USA detail data: here it pays that ethnicgroup, religion, and language are subclasses of "PercentageProperty".
- Digester: The Digester is a mixture of hand-made in-memory model and Stream-Reading. The problem with Calexit is that a complete output must be created. So everything must be stored in-memory in Java classes – how to generate the XML output then? The result would be a hand-made coding of JAXB-style unmarshalling, applying the respective changes, and another hand-made serialization as XML again.

```
package JAXBmondial;
```
```

```
import javax.xml.bind.JAXBContext;
import javax.xml.bind.Unmarshaller;
import javax.xml.bind.Marshaller;
import java.io.File;
import java.util.List;
import java.util.LinkedList;
import java.util.GregorianCalendar;
import javax.xml.datatype.XMLGregorianCalendar;
import javax.xml.datatype.DatatypeFactory;
import java.math.BigDecimal;
import java.math.BigInteger;
import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.xpath.XPathExpression;
import org.jdom2.xpath.XPathFactory;
import org.jdom2.input.SAXBuilder;

public class CalexitJAXB {
 static Province calprov = null;
 static Country usa = null;
 static Country california = null;

 static BigInteger calpop = null;
 static BigInteger uspop = null;
 static BigDecimal calGDP = null;
 static BigDecimal usGDP = null;

 public static void main (String args[]) {
 try {
  JAXBContext jc = JAXBContext.newInstance("JAXBmondial");
  Unmarshaller unmarshaller = jc.createUnmarshaller();

  Mondial mondial = (Mondial) unmarshaller.unmarshal(new File("mondial.xml"));
  List<Country> countrylist  = mondial.getCountry();
  california = (Country) unmarshaller.unmarshal(new File("caldata.xml"));
    // the borders/country are set to null because id are not there
  california.getBorder().clear(); // must get them from a separate DOM:

  SAXBuilder builder = new SAXBuilder();
  File caldataFile = new File("caldata.xml");
  Document caldataxml = builder.build(caldataFile);

  int index;
  boolean found = false;
  for ( int i = 0; i < countrylist.size() && !found ; i++ )
  {
    Country country  = countrylist.get(i);
    if ("USA".equals(country.getCarCode())) {
      usa = country;
      mondial.getCountry().add(i+1,california);
      List<Province> usprovlist  = country.getProvince();
      for ( int j = 0; j < usprovlist.size() && !found ;j++ ) {
        Province usprov = (Province) usprovlist.get(j);
        if ("California".equals(usprov.getName().get(0))) {
            calprov = usprov;
            usprovlist.remove(j);
            found = true;
         }
       }
    }
  }
  String calID = calprov.getId();
  california.setCarCode("CA");
  BigDecimal calarea = calprov.getArea();
  california.setArea(calarea);
```

```
List<Object> calmembers = california.getMemberships();
for (Object org : usa.getMemberships()) {
  if (!"Group␣of␣5".equals(((Organization)org).getName())
      && !"Group␣of␣7".equals(((Organization)org).getName())) {
    calmembers.add(org);
    for (Members ms : ((Organization)org).getMembers()) {
      List<Country> memberslist = new LinkedList<Country>();
      for (Object x: ms.getCountry())
        memberslist.add((Country)x); // avoid ConcModException
      for (Country x : memberslist)
         if (x == usa)
            ms.getCountry().add(california);
    }
  }
}
// put CAL name, populations and cities into cal country
for (String name:calprov.getName())
  california.getName().add(name);
List<Population> calpops = california.getPopulation();
for (Object pop : calprov.getPopulation())
  calpops.add((Population)pop);
List<City> calcities = california.getCity();
for (City city : calprov.getCity()) {
  calcities.add(city);
  city.setCountry(california);
  city.setProvince(null);
  if (city == calprov.getCapital())
    city.setIsCountryCap("yes");
}
// borders of neighbors and USA-neighbors-borders
// must be taken from the CALdata-DOM (country refs are empty in the JAXB):
XPathFactory xpf = XPathFactory.instance();
XPathExpression xpath = xpf.compile("//border[@country='USA']");
Element e = (Element) xpath.evaluateFirst(caldataxml);
// BigDecimal has no valueOf from String
BigDecimal uscallength = new BigDecimal(e.getAttributeValue("length"));
xpath = xpf.compile("//border");
List calxmlborders = xpath.evaluate(caldataxml);
for (Object calborderel: calxmlborders) {
  String othercode = ((Element)calborderel).getAttributeValue("country");
  BigDecimal len =
      new BigDecimal(((Element)calborderel).getAttributeValue("length"));
  if (!"USA".equals(othercode)) {
    // ugly: find the other country in the JDOM Tree :((
    // No XPath possible!
    // Iterate over all countries to search which ones have to be updated?
    // => no: iterate over US borders and use their references to the US neighbors!
    //    (all CA neighbors must also be current USA neighbors)
    List<Border> usaborders = usa.getBorder();
    for (Border usaborder : usaborders) {
      Country othercountry = (Country)usaborder.getCountry();
      if (othercode.equals(othercountry.getCarCode())) {
        List<Border> otherborders = othercountry.getBorder();
        for (Border otherborder : otherborders) {
          if (otherborder.getCountry().equals(usa)) {
            BigDecimal oldborderlength = otherborder.getLength();
            BigDecimal newborderlength = oldborderlength.subtract(len);
            otherborder.setLength(newborderlength);
            usaborder.setLength(newborderlength);
            Border calotherborder = new Border();
            calotherborder.setCountry(othercountry);
            calotherborder.setLength(len);
            california.getBorder().add(calotherborder);
          }
        }
```

```
        Border othercalborder = new Border();
        othercalborder.setCountry(california);
        othercalborder.setLength(len);
        othercountry.getBorder().add(othercalborder);
      }
    }
  }
  else {
    Border uscalborder = new Border();
    uscalborder.setCountry(california);
    uscalborder.setLength(len);
    usa.getBorder().add(uscalborder);
    Border calusborder = new Border();
    calusborder.setCountry(usa);
    calusborder.setLength(len);
    california.getBorder().add(calusborder);
  }
}
// geo things located in CAL ... inverse refs are not supported
// ... thus, each of the mondial.getGeo() lists has to be traversed
// ... there is even no common superclass "Geo" available
//
for (Mountain x:mondial.getMountain()) {
    if (x.getCountry().contains(usa)) {
        processlocateds(x.getLocated(), x.getCountry());
    }
}
for (Island x:mondial.getIsland()) {
    if (x.getCountry().contains(usa)) {
        processlocateds(x.getLocated(), x.getCountry());
    }
}
for (River x:mondial.getRiver()) {
    if (x.getCountry().contains(usa)) {
        processlocateds(x.getLocated(), x.getCountry());
        processlocateds(x.getSource().getLocated(), x.getSource().getCountry());
        processlocateds(x.getEstuary().getLocated(), x.getEstuary().getCountry());
    }
}
for (Lake x:mondial.getLake()) {
    if (x.getCountry().contains(usa)) {
        processlocateds(x.getLocated(), x.getCountry());
    }
}
for (Sea x:mondial.getSea()) {
    if (x.getCountry().contains(usa)) {
        processlocateds(x.getLocated(), x.getCountry());
    }
}
for (Desert x:mondial.getDesert()) {
    if (x.getCountry().contains(usa)) {
        processlocateds(x.getLocated(), x.getCountry());
    }
}

// details: update USA country data
// area attribute
usa.setArea(usa.getArea().subtract(calarea));
for (Encompassed enc: usa.getEncompassed()) {
  Encompassed calenc = new Encompassed();
  calenc.setContinent(enc.getContinent());
  calenc.setPercentage(enc.getPercentage());
  california.getEncompassed().add(calenc);
}
california.setGovernment("democracy");
```

```
 IndepDate calindep = new IndepDate();
 calindep.setFrom(usa.getCarCode());
 calindep.setValue(DatatypeFactory.newInstance().
                   newXMLGregorianCalendar(new GregorianCalendar(2016, 12, 16)));
 // populations
 List<Population> uspoplist = new LinkedList<Population>();
 for (Population pop: usa.getPopulation()) {
      uspoplist.add(pop); // avoid ConcModException
      uspop = pop.getValue(); // and keep the last one as uspop
 }
 for (Population pop: uspoplist) {
   found = false;
   for (Population cpop: california.getPopulation()) {
       if (cpop.getYear() == pop.getYear()) {
         pop.setValue(pop.getValue().subtract(cpop.getValue()));
         found = true;
         calpop = cpop.getValue(); // keeps the last one as calpop
       }
   }
   if (!found) usa.getPopulation().remove(pop);
 }
 // weigh by population:
 usa.setPopulationGrowth(weighByPop(usa.getPopulationGrowth(),
                                    california.getPopulationGrowth()));
 usa.setInfantMortality(weighByPop(usa.getInfantMortality(),
                                   california.getInfantMortality()));
 usa.setUnemployment(weighByPop(usa.getUnemployment(),
                                california.getUnemployment()));

 // GDPs
 usGDP = usa.getGdpTotal();
 calGDP = california.getGdpTotal();
 usa.setGdpTotal(usGDP.subtract(calGDP));
 usa.setGdpAgri(weighByGDP(usa.getGdpAgri(), california.getGdpAgri()));
 usa.setGdpInd(weighByGDP(usa.getGdpInd(), california.getGdpInd()));
 usa.setGdpServ(weighByGDP(usa.getGdpServ(), california.getGdpServ()));
 usa.setInflation(weighByGDP(usa.getInflation(), california.getInflation()));

 // Ethnicgrps, religions, languages  /*
 handleByGroups(usa.getEthnicgroup(), california.getEthnicgroup());
 handleByGroups(usa.getReligion(), california.getReligion());
 handleByGroups(usa.getLanguage(), california.getLanguage());

 // output to a file:
 File outfile = new File("outputjaxb.xml");
 //JAXBContext jc = JAXBContext.newInstance(Mondial.class);  -- already defined above
 Marshaller m = jc.createMarshaller();
 m.setProperty("com.sun.xml.bind.xmlHeaders", "\n<!DOCTYPE␣mondial␣SYSTEM␣'mondial.dtd'>");
 m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true); // if set to true -> validation complains
 m.setProperty("com.sun.xml.bind.indentString", "␣␣"); // indentation only by 2 (default 4)
 m.marshal(mondial, outfile);
 } catch (Exception e ) { e.printStackTrace(); }
}

// tedious type conversions because JAXB assigns hard-to-use datatypes
private static BigDecimal weighByPop(BigDecimal usvalue, BigDecimal calvalue) {
  return ((usvalue.multiply(new BigDecimal(uspop)))
          .subtract(calvalue.multiply(new BigDecimal(calpop))))
    .divide(new BigDecimal(uspop.subtract(calpop)),
            java.math.BigDecimal.ROUND_HALF_UP);
}

private static BigDecimal weighByGDP(BigDecimal usvalue, BigDecimal calvalue) {
  return((usvalue.multiply(usGDP)).subtract(calvalue.multiply(calGDP)))
    .divide(usGDP.subtract(calGDP), java.math.BigDecimal.ROUND_HALF_UP);
```

```
 }

 // exploit that EthnicGroup, Religion, and Language are subclasses
 //  of the class "PercentageProperty"
private static void handleByGroups(List<PercentageProperty> uslist,
                                   List<PercentageProperty> callist) {
   for (PercentageProperty pus : uslist) {
     for (PercentageProperty pcal : callist) {
       if (pus.getValue().equals(pcal.getValue())) {
         pus.setPercentage((pus.getPercentage().multiply(new BigDecimal(uspop))
                           .subtract(pcal.getPercentage()
                                     .multiply(new BigDecimal(calpop))))
                            .divide(new BigDecimal(uspop.subtract(calpop)),
                                    java.math.BigDecimal.ROUND_HALF_UP));
       }
     }
   }
 }

 private static void processlocateds(List<Located> locs,
                                     List<Object> countries) {
   Boolean incal = false;
   Boolean inother = false;
   Located usloc = null;
   for (Located loc:locs) {
     if (usa.equals(loc.getCountry())) {
       usloc = loc;
       for (Object prov:loc.getProvince()) {
         if (prov == calprov) incal = true;
         else inother = true;
       }
     }
   }
   if (incal) {
     usloc.getProvince().remove(calprov);
     if (inother) countries.add(california);
     else { //  if (incal && !inother) {
       countries.add(california);
       countries.remove(usa);
       locs.remove(usloc);
       usloc.getProvince().remove(calprov);
     }
   }
 }
}
```

Comparison of the Solutions and their problems:

XSLT: perfect. A simple rule for "copy everything" and overriding it with more specific rules.

- little bit tricky do deal with IDREFS modification/construction.
- output formatting/indentation built-in.

JDOM: basically, simple, boring programming on the XML storage structure.

- No IDREF functionality of reference attributes (could be there, since DTD is! - a simple ID dictionary would do it)
  ⇒ if one needs it, create the dictionary by a recursive walk.
- Working with lists (explicit iteration necessary + adding/removing) requires copying to avoid Concurrent Modification Exceptions.
- Text contents and attribute values are always strings. No way of simple XQuery/XSLT-like casting. Datatypes and conversion methods have to be applied extensively (at least, the user can decide which java classes to use – converting everything as Float does it) [same as for

SAX/StAX].

- flexible and useful output/serialization functionality from DOM/JDOM.

StAX: (applies also to SAX) Stream-based on-the-fly processing

- Requires a strategy for buffering/keeping fragments. A simple choice would be to use manually created DOM fragments. This requires (i) rules for all elements that have to be cached, and (ii) manual serialization afterwards into the output stream!. Using a BufferedStream instead, and just setting the writer on it saves these efforts, and also has the advantage of being technically consistent with the rest.
- Using a two-pass-strategy may be useful in general (1st pass: collect all necessary information, 2nd pass: modify data).
- Text contents and attribute values are always strings. No way of simple XQuery/XSLT-like casting. Datatypes and conversion methods have to be applied extensively (at least, the user can decide which java classes to use – converting everything as Float does it) [same as for DOM].
- Implicit whitespace from formatting tends to be ugly (in=out in general, but for new elements, the indentation would require some work).

JAXB: basically, the task is not much different from DOM: Simple, boring programming; now on a derived object-oriented storage structure.

- Requires to have an XML Schema.
- IDREFS are smoothly turned into object references (and back).
- Numerical datatypes are dictated by JAXB. It chooses BigInteger, BigDecimal, ... which are incompatible and require heavy application of conversion methods, and use method syntax for arithmetics.
- Working with lists (explicit iteration necessary + adding/removing) requires copying to avoid Concurrent Modification Exceptions.
- Flexible and useful output/serialization functionality.