XML
ECE 155B – Spring 2012
Discussion Section 1.1

What is XML?

- XML stands for Extensible Markup Language
- XML is a ***markup language*** like html
- XML was designed to ***describe data***
- XML tags are not predefined; you must ***define your own tags***
- XML uses a ***Document Type Definition*** (***DTF***) or an ***XML Schema*** to describe the data
- XML with a DTD or XML Schema is designed to ***self-descriptive***

The Main Differences between XML and HTML

- XML is not a replacement for HTML
- XML and HTML were designed with different goals:
    - XML: Describe the data and focus on what the data is
    - HTML: Display the data and focus on how it looks
- HTML is about displaying information, while XML is about describing information
- XML does not DO anything
- XML was not designed to DO anything

Maybe it is hard to understand, but XML does not DO anything. XML was created to structure, store and send information.

The following example is a note to Tove from Jani, stored as XML:

```
<note>
<to>Tove</to>
<from> Jani </from>
<header> Reminder </ header >
<body> Don't forget your TA's birth date! </body>
</note>
```

The note has a header and a message body. It also has sender and receiver information. But, this XML document does not DO anything. It is just pure information wrapped in XML tags. Someone must write a piece of software to send, receive and display it.

- XML is free and extensible
- XML allows you to define your own tags and structure of XML documents. XML tags are not predefined; you must "invent" your own tags. The tags in the example above (like <to> and <from>) are not defined by the XML standard. These tags were "invented" by the author of the XML document. In contrast, tags and the structure of HTML documents are predefined by the HTML standard. The author of an HTML document can use only tags that are predefined in the HTML standard, such as <p>, <h1>, etc.
- XML is a complement to HTML; XML is not a replacement for HTML. In future Web development, it is most likely that XML will be used to describe the data, while HTML will be used to format and display the same data.

- XML is a cross-platform, software and hardware independent language for transmitting information.

XML in Future Web Development

- XML is going to be everywhere

We have been participating in XML development since its creation. It has been amazing to see how quickly the XML standard has been developed and how quickly a large number of software vendors have adopted it.

We strongly believe that XML will be as important to the Web of the future as HTML has been to the Web of the past and that XML will be the most common format for data manipulation and transmission.

What Does XML Look Like?

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to> Yung-Ting  </to>
<from> Professor Moser </from>
<heading> Discussion Reminder[April 1]</heading>
<body>Don't forget the discussion on Friday at 1pm, Yung-Ting  </body>
</note>
```

1) XML uses a self-describing and simple syntax
2) XML elements have opening and closing tags
3) The first line of an XML file describes the XML version. We don't need to worry about this, just assume that it's <?xml version="1.0" encoding="ISO-8859-1"?>
4) XML tags are **case-sensitive**
5) XML tags must be **properly nested**

Example:

```
<email>
<to>Yung-Ting</to>
<from>Professor Moser</from>
<body>
<message>Your discussion notes need some work!</message>
<signature>--Professor Moser
</body>
</signature>
```

The above example is invalid because the <body> tag is closed before the <signature> tag, but the <signature> tag is opened after the <body> tag. This is analogous to a stack, opening tags are pushed on the stack, and closing tags are popped off the top of the stack.

6) *XML attributes* are optional parts of an opening tag

Example:

```
<user name="Yung-Ting" major="CE">TA</user>
```

How Do You Write in XML Format?

When we are to Patient information, we will use the toXML method to write the information in XML format. This method will be similar to the *toString* method, but will return the String in XML format.

Example code for Patient might look like:

Doctor class

```
public class Doctor
{
    public String name = "MyName";
    public String lastname = "MyLastName";

    Vector <Patient> patients; // doctor keep patient information in a vector

    public String toXML()
    {
        String xml = "<Doctor>";
        xml += "<DoctorName>"+name+"</DoctorName>";
        xml += "<DoctorLastName>"+lastname+"</DoctorLastName>";
        xml += "<Patients>";
        for (int i = 0; i < patients.size(); i++)
            xml += patients.elementAt(i).toXML(); // add xml code for each patient
        xml += "</Patients>";
        xml += "</Doctor>";

        return xml;
    }
}
```

The main XML method for writing to a file

```
  write ("<?xml version="1.0" ?>");
  write (doctor.toXML());
```

Other class members such as Patient and Note will be completed similarly.

How Do You Read an XML File?

To read XML files, we will use import javax.xml.parsers and org.xml.sax packages.

```
SAXParserFactory spf = SAXParserFactory.newInstance()
SAXParser sp = spf.newSAXParser()
```

Here we create a parser object. We need to provide a parser for a HandlerBase object which has the following methods for us to overwrite.

void startElement(*String name, AttributeList list*)
void endElement(*String name*)

When the parser recognizes a tag opening, it calls the startElement method of the handler and, similarly, the endElement for a tag closing. If there are attributes, this will be provided in the attribute list. For our project we don't need this, we will just use the *name* parameter. Our implementation handler has a formal parameter of Vector type that stores Patient objects.

```
XMLParseHandler handler = new XMLParseHandler()
sp.parse ("c:\\doctor.xml",handler)
Doctor doctor = handler.doctor;
```

With the handler, we can just call the parse method with the *filename* and the *handler* object. Here is the XMLParseHandler code:

```
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import javax.xml.parsers.*;

public class XMLParser extends DefaultHandler
{
   public Doctor doctor;
   private Patient patient;       // temporary object
   private Note note;             // temporary object
   private StringBuffer accumulator = new StringBuffer(); // Buffer used during parsing

   public XMLParser()
   {
       doctor = new Doctor();
   }
   public void characters(char[] buffer, int start, int length)
   {
      accumulator.append(buffer, start, length);
   }
   // - - - - - - - - - - - - - - - - - - - -
   // Each time a new tag is opened, this method is called
   public void startElement(String uri,String lname, String name, Attributes attributes)
   {
      // - - - - - - - - - - - - - - - - - - - -
      // New Note starts
      if(name.equals ("Note"))
      {
         note = new Note();          // we will set values of this later
         patient.addNote(note);      // add to patient
      }
```

```
        // - - - - - - - - - - - - - - - - - - - - -
        // New Patient starts
        else if(name.equals ("Patient"))
        {
           patient = new Patient();       // we will set values of patient later
           doctor.addPatient(patient);  // add to doctor
        }
        // - - - - - - - - - - - - - - - - - - - - -
        // Reset accumulator
        accumulator.setLength(0);
    }
    public void endElement(String uri,String lname,String name)
    {
        String value = accumulator.toString();
        // - - - - - - - - - - - - - - - - - - - - -
        //    Doctor information
        if(name.equals ("DoctorName"))
           doctor.name= value;
        else if(name.equals ("DoctorLastName"))
           doctor.lastname = value;
        // - - - - - - - - - - - - - - - - - - - - -
        //    Patient information
        else if(name.equalsIgnoreCase("PatientSSN"))
           patient.ssn = value;
        else if(name.equalsIgnoreCase("PatientFirstName"))
           patient.name = value;
        // - - - - - - - - - - - - - - - - - - - - -
        //    Note information
        else if(name.equalsIgnoreCase("Author"))
           note.author = value;
        else if(name.equalsIgnoreCase("Date"))
           note.date = value;
    }
}
```

In the ***endElement*** method, we set the values of Objects. If it is the object name such as Patient, Note, Doctor we do not need to do anything about them.

If you have more formal parameters than these, you will need to modify the code accordingly.

Warning:

It is important that the tag names that you use when writing and when parsing are the same. Otherwise, you will have trouble reading what you have written. This is important not only at this point but also for server-client interactions as we will discover in later assignments.

Links
More on XML parsing with Java
http://www.oreilly.com/catalog/jenut2/chapter/ch19.html

Example of the xml file you will build for project1:

```
<?xml version='1.0' ?>
<Doctor>
   <DoctorName>Yung-Ting</DoctorName>
   <DoctorLastName>Chuang</DoctorLastName>
   <Patients>
      <Patient>
         <PatientSSN>123-45-2355</PatientSSN>
         <PatientFirstName>Po-Yo</PatientFirstName>
         <Notes></Notes>
      </Patient>
      <Patient>
         <PatientSSN>236-34-0292</PatientSSN>
         <PatientFirstName>Shu-Ha</PatientFirstName>
         <Notes>
            <Note>
               <Author>Yung-Ting Chuang</Author>
               <Date>2011/3/24</Date>
               <Content>Has foot ingury</Content>
            </Note>
         </Notes>
      </Patient>
   </Patients>
</Doctor>
```