COMP S380F Web Applications: Design and Development
Lab 2: XML, JSON, Servlet

In this lab session, we will learn two common data exchange standards: ***XML*** and ***JSON***. We will also use the ***NetBeans IDE 8*** to create a simple ***Servlet*** and run it on the Web container ***Apache Tomcat***.

Task 1: Data exchange standards XML and JSON

There are different machines and platforms working together over the Internet. Data exchange is one of the basic requirements of the Internet. We must have a standard to exchange data. XML and JSON are two commonly used data formats.

XML

XML (eXtensible Markup Language) is developed by the W3C and has the following properties:

- XML is a markup language like HTML. But XML was designed to ***carry data***, while HTML was designed to display data.
- XML was designed to be ***self-descriptive***; XML is just information wrapped in markup tags.
- XML was designed to be ***extensible***. XML does not use predefined tags. Users can define their own terms and markups, and add more terms and markups to carry more information.

Below is an example of a XML document. Data is marked up by self-describing tags.

```
<?xml version="1.0" encoding="UTF-8"?>
<breakfast_menu>
    <food>
        <name>Belgian Waffles</name>
        <price>$5.95</price>
        <description>Our famous Belgian Waffles with plenty of real maple syrup</description>
        <calories>650</calories>
    </food>
    <food>
        <name>French Toast</name>
        <price>$4.50</price>
        <description>Thick slices made from our homemade sourdough bread</description>
        <calories>600</calories>
    </food>
    <food>
        <name>Homestyle Breakfast</name>
        <price>$6.95</price>
        <description>Two eggs, bacon or sausage, toast, and our ever-popular hash browns</description>
        <calories>950</calories>
    </food>
</breakfast_menu>
```

Each XML document is formed as an **element tree**. In the above example, the **root element** is “breakfast_menu”. Each element can have **child elements**, e.g., “breakfast_menu” has three child elements “food”, which in turn has four child elements “name”, “price”, “description” and “calories”.

The syntax rules of XML are very simple and logical. Below are some important rules to note:

1. XML documents must contain a ***single root element*** that is the ancestor of all other elements.
2. The XML prolog (`<?xml version="1.0" encoding="UTF-8"?>`) is optional. If it exists, it must come first in the document.

3. All XML elements must have an end tag. For example,
   `<p>paragraph</p>` (Here, "paragraph" is the **text content** of the p element.)
   `<br />`
4. XML tags are ***case-sensitive***. The start and end tags must be written with the same case.
5. XML elements must be properly nested, so two different XML elements may either have an ancestor-descendant relationship or be independent.
6. XML elements can have ***attributes*** in name-value pairs, where the values must always be quoted.

JSON

JSON (JavaScript Object Notation) is a light-weight alternative to XML for data exchange. It is a language-independent data format, derived from JavaScript. It is described by RFC 7159. (RFC stands for Requests for Comments. An RFC document contains technical and organizational notes about the Internet.) Like XML, JSON is ***self-describing*** and easy to understand.

A JSON document is a collection of name-value pairs, as shown in the following example.

```
{"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
]}
```

The JSON format is syntactically identical to the code for creating JavaScript objects. Its syntax includes:

1. Data is in ***name-value pairs***.
   - ***JSON names require double quotes***, while JavaScript names don't.
   - JSON values can be a number (integer or floating point), a string (in double quotes), a Boolean (true or false), an array (in square brackets), an object (in curly braces), null.
2. Data is separated by commas.
3. Curly braces hold ***objects***. Each JSON object can contain multiple name-value pairs, separated by commas. E.g., `{"firstName":"John", "lastName":"Doe"}`.
4. Square brackets hold ***arrays***. In the above example, the value of "employees" is an array containing three objects; each of which is a record of a person (with a first name and a last name).

Difference between JSON and XML

For presenting the same information, JSON is shorter and is quicker to read and write than XML. Thus, JSON is more light-weight than XML. Also, JSON doesn't use end tag and allows the use of arrays.

Due to the similarity of the JSON format and JavaScript objects, a JavaScript program can use standard JavaScript functions to convert JSON data into native JavaScript objects. However, a JavaScript program must parse an XML document using an XML parser to parse the XML document as an element tree.

Your task: You are given the HTML file task1.html, which can convert an XML document into JSON format using JavaScript. (It is adapted from the webpage https://davidwalsh.name/convert-xml-json.) Use it to convert some XML files into JSON format and try to understand the differences between the input XML and the output JSON. Three sample XML files are provided for you.

Task 2: Creating a simple Servlet using NetBeans 8 & Running it on Apache Tomcat

A Servlet is a Java program that runs within a Web server. Servlets receive and respond to requests from Web clients, usually across HTTP. In this task, we will use NetBeans IDE 8 to create a simple Servlet called "helloServlet". We will run the Servlet on the web container Apache Tomcat and then use NetBeans to debug the Servlet.

Your task: Upon receiving an HTTP GET request, the Servlet "helloServlet" will return the string "Hello, World!" in the HTTP response. Follow the following steps:

1. In NetBeans 8, create a new project with the following properties:
   - Category: ***Maven***
   - Project: ***Web Application***
   - Project name: Hello-World
   - Group Id: ouhk.comps380f
   - Package: ouhk.comps380f
   - Server: Apache Tomcat
   - Java EE Version: Java EE 7 Web

2. In the "Projects" section, select **Project Files > pom.xml**. (POM stands for "Project Object Model" and pom.xml is a one-stop-shop for all things concerning the Maven project.)

   In pom.xml, look for the XML tag ***<dependencies>***, which is the project's dependency list. Most project depends upon other libraries to build and run correctly. Maven downloads and links the dependencies on compilation, including the dependencies of those dependencies (transitive dependencies), allowing the dependency list to focus solely on the dependencies the project requires.

   Update the <dependencies> tag, as follows. Then, right-click on the project name and select "***Build with Dependencies***" to let NetBeans download the required library from Maven.

```
<dependencies>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>3.1.0</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

3. In the "Projects > Source Packages" section, right-click on Java package **ouhk.comps380f**. Select **New > Servlet…** to create a Servlet with the following properties:
   - Class Name: HelloServlet
   - Check the checkbox "***Add information to deployment descriptor (web.xml)***"
   - Servlet Name: helloServlet
   - URL Pattern(s): /greeting

4. In the Servlet HelloServlet.java, modify the ***doGet()*** method, as follows:

```
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    response.getWriter().println("Hello, World!");
}
```

5. In the "Projects > Web Pages" section, select the **deployment descriptor /WEB-INF/web.xml**. Add a ***<display-name>*** element and modify the ***<servlet-mapping>*** element, as follows:

```
<display-name>Hello World Application</display-name>

<servlet>
   <servlet-name>helloServlet</servlet-name>
   <servlet-class>ouhk.comps380f.HelloServlet</servlet-class>
</servlet>
<servlet-mapping>
   <servlet-name>helloServlet</servlet-name>
   <url-pattern>/greeting</url-pattern>
   <url-pattern>/salutation</url-pattern>
   <url-pattern>/wazzup</url-pattern>
</servlet-mapping>
```

6. Every time when you update some class in your web application, you need to build the web application again using "***Build with Dependencies***". Then, you can run the project (press F6) on Apache Tomcat. A web browser will be opened and display the default index page of the project at http://localhost:8084/Hello-World/. Note that this is ***not*** the output returned by helloServlet.

7. To see the output from helloServlet, you can try any one of the following URLs:
   - http://localhost:8084/Hello-World/greeting
   - http://localhost:8084/Hello-World/salutation
   - http://localhost:8084/Hello-World/wazzup

8. To **understand the life cycle of a Servlet**, add the ***init()*** and ***destroy()*** methods, and run again.

```
@Override
public void init() throws ServletException {
    System.out.println("Servlet " + this.getServletName() + " has started.");
}

@Override
public void destroy() {
    System.out.println("Servlet " + this.getServletName() + " has stopped.");
}
```

To debug the Servlet, we can use the debugger provided by NetBeans. Follow the following steps to try using this debugger:

1. Create breakpoints: In HelloServlet.java, click on the line number of each of the Java statements inside the three functions doGet(), init() and destroy() to create breakpoints (with red square).
2. Debug the project (press Ctrl+F5) and follow Step 7 above. Notice the output and how the debugger interrupts the execution of helloServlet (You can press F5 to continue the execution).