Session 14

Serialization/JSON

Lecture Objectives

- Understand the need for serialization
- Understand various approaches to serialization
- Understand the use of JSON as a popular approach to serialization
- Understand how to access JSON data from JavaScript and Java
How Do We Transmit Objects Between Servers?

- We previously covered some data transmission approaches
  - Primitives (e.g., form data set name/value pairs)
  - Specific structured data (e.g., JPEG image) as a MIME data type
- But many objects involve structured data that is not logically represented as a stream

- Approaches
  - Java Serialization
  - XML
  - JSON

Most examples in this set of slides are taken from W3Schools tutorial
Terminology

- **Serialization**
  - Process of translating data structures or object state into a format that can be stored and reconstructed later in the same or another computer environment (also called *marshalling*)
  - A simple way to persist live objects to persistent storage
- **Unmarshalling** - reverse process

Java Serialization

- java.io.Serializable interface - must be declared
- java.io.Externalizable interface
  - writeExternal method
  - readExternal method
- Platform independent (serialized on one platform, reconstructed on another platform)
- No serialization methods declared on the Serializable Interface
package lectures;
public class Employee implements java.io.Serializable {
    public String name;
    public String address;
    public transient int SSN;
    public int number;
    
    public void mailCheck() {
        System.out.println("Mailing a check to " + name + " " + address);
    }
}

All fields of a serialized class must be declared Serializable or transient (not serialized)

Example from tutorialspoint.com

public class SerializeDemo {
    public static void main(String[] args) {
        Employee e = new Employee();
        e.name = "Reyan Ali";
        e.address = "Phokka Kuan, Ambehta Peer";
        e.SSN = 11122333;
        e.number = 101;
        try {
            FileOutputStream fileOut = new FileOutputStream("employee.ser");
            ObjectOutputStream out = new ObjectOutputStream(fileOut);
            out.writeObject(e);
            out.close();
            fileOut.close();
            System.out.printf("Serialized data is saved in employee.ser");
        } catch(IOException i) {
            i.printStackTrace();
        }
    }
}
... Java Serialization Example

```java
public class DeSerializeDemo {
    public static void main(String[] args) {
        Employee e = null;
        try {
            FileInputStream fileIn = new FileInputStream("employee.ser");
            ObjectInputStream in = new ObjectInputStream(fileIn);
            e = (Employee) in.readObject();
            in.close();
            fileIn.close();
        } catch (IOException i) {
            i.printStackTrace();
            return;
        } catch (ClassNotFoundException c) {
            System.out.println("Employee class not found");
            c.printStackTrace();
            return;
        }
        System.out.println("Deserialized Employee...");
        System.out.println("Name: " + e.name);
        System.out.println("Address: " + e.address);
        System.out.println("SSN: " + e.SSN);
        System.out.println("Number: " + e.number);
    }
}
```

SSN was declared transient. When the object is recreated the default value for a transient int is 0
Uses of Java Serialization

- Persisting objects to be reused in the same or similar environment
- Not useful for sharing objects with non-Java environments
- Alternatives
  - XML
  - JSON

What is JSON?

- JavaScript Object Notation
- Data serialization format
- Open standard format for the interchange of name/value pair objects
- Alternative to XML
- Language independent format, although originally derived from JavaScript
How Do You Pronounce JSON?

- It doesn’t matter (according to the inventor)
- The way your colleagues pronounce it
  - Just like the name (Jason) or
  - Jay-Sahn

Background

- The JSON format is syntactically identical to the code for creating JavaScript objects
- Unlike XML, you don’t need an external parser
- JavaScript function available to convert JSON data into a native JavaScript object
- Very useful in sharing data with a browser client
Revisit JavaScript

- Objects
  - Unordered collection of properties
  - Each property has a name and a value
  - Property names are strings
- Examples
  - {} - empty
  - {x:0, y:0}

Remember, JavaScript functions are objects

Note the use of quotes in a JavaScript literal when the name includes spaces

Easy to define a new object

```
var position = {x:0, y:0};
```

Arrays

- Arrays
  - Order collection of values
  - Untyped
  - Array elements may be objects or other arrays
Example

Code below shows parsing of JSON text data

```html
<!DOCTYPE html>
<html>
<body>
<h2>JSON Object Creation in JavaScript</h2>
<p id="demo"> </p>
<script>
 var text = '{"name":"John Johnson", "street":"Oslo West 16", "phone":"555 1234567"}';
 var obj = JSON.parse(text);
 document.getElementById("demo").innerHTML = 
  obj.name + "<br>" +
  obj.street + "<br>" +
  obj.phone;
</script>
</body>
</html>
```

XML / JSON Comparison

```json
{"menu": {
 "id": "file",
 "value": "File",
 "popup": {
 "menuitem": [
 {"value": "New", "onclick": "CreateNewDoc()"},
 {"value": "Open", "onclick": "OpenDoc()"},
 {"value": "Close", "onclick": "CloseDoc()"}
 ]
}
}}
```

The same text expressed as XML:

```xml
<menu id="file" value="File">
 <popup>
  <menuitem value="New" onclick="CreateNewDoc()" />
  <menuitem value="Open" onclick="OpenDoc()" />
  <menuitem value="Close" onclick="CloseDoc()" />
 </popup>
</menu>
```
XML / JSON Comparison

- Both XML and JSON are
  - Self describing
  - Hierarchal
  - Can be fetched with an XMLHttpRequest
- parse is a JavaScript function
- XML requires clumsier access
  - external parser
  - temporary variables for the parsed results
  - Tree traversal

JSON Syntax

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays
- JSON names require double quotes

JSON syntax and JavaScript literal syntax are closely related, but not exactly the same.
**JSON Values**

- 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

---

**Accessing JavaScript Object Data**

```javascript
var employees = [
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter","lastName": "Jones"}
];
// returns John Doe
employees[0].firstName + " " + employees[0].lastName;

Employees is an array of objects and
firstName is a property of an element of
the array
```
Storing and Retrieving from localStorage

```html
<!DOCTYPE html>
<html>
<body>
<h2>Store and retrieve data from local storage.</h2>
<p id="demo"></p>
<script>
var myObj, myJSON, text, obj;
myObj = { "name": "John", "age": 31, "city": "New York" };
myJSON = JSON.stringify(myObj);
localStorage.setItem("testJSON", myJSON);
text = localStorage.getItem("testJSON");
obj = JSON.parse(text);
document.getElementById("demo").innerHTML = obj.name;
</script>
</body>
</html>
```

localStorage is a property of the window object. Browsers write text to localStorage. The `stringify` and `parse` methods perform marshalling and unmarshalling of JavaScript objects.

JSON Syntax

- JSON format is almost identical to that of JavaScript objects.
- Keys must be strings, written with double quotes (JavaScript allows strings, numbers or identifiers).
- JSON values must be one of:
  - string
  - number
  - object
  - array
  - boolean
  - null

Note that functions are not valid JSON values.
Example - Code

```html
<div id="id01"></div>
<script>
var xmlhttp = new XMLHttpRequest();
var url = "myTutorials.txt";

xmlhttp.onreadystatechange = function() {
    if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
        var myArr = JSON.parse(xmlhttp.responseText);
        myFunction(myArr);
    }
}
xmlhttp.open("GET", url, true);
xmlhttp.send();

function myFunction(arr) {
    var out = "";
    var i;
    for(i = 0; i < arr.length; i++) {
        out += '<a href="' + arr[i].url + '">' + arr[i].display + '</a><br>';
    }
    document.getElementById("id01").innerHTML = out;
} </script>

Out is used to build the HTML that is inserted into the div block.

myTutorials.txt - JSON Data

[ { "display": "HTML Tutorial", "url": "http://www.w3schools.com/html/default.asp" },
  { "display": "CSS Tutorial", "url": "http://www.w3schools.com/css/default.asp" },
  { "display": "JavaScript Tutorial", "url": "http://www.w3schools.com/js/default.asp" },
  { "display": "jQuery Tutorial", "url": "http://www.w3schools.com/jquery/default.asp" },
  { "display": "JSON Tutorial", "url": "http://www.w3schools.com/json/default.asp" },
  { "display": "AJAX Tutorial", "url": "http://www.w3schools.com/ajax/default.asp" },
  { "display": "SQL Tutorial", "url": "http://www.w3schools.com/sql/default.asp" },
  { "display": "PHP Tutorial", "url": "http://www.w3schools.com/php/default.asp" },
  { "display": "XML Tutorial", "url": "http://www.w3schools.com/xml/default.asp" }]

File on CSE336 Web site
Are We on Track?

Modify and run the example so that
- The tutorial names do not contain an anchor tag
- The names appear in an unordered list

Steps (to get around the Same Origin Policy)
- Download the example html
- Download the myTutorials.txt file
- Insert both files into your NetBeans project
- Modify the JavaScript in the html file

function myFunction(arr) {
    var out = "<ul>
    var i;
    for(i = 0; i < arr.length; i++) {
        out += "<li>" + arr[i].display + "</li>";
    }
    out += "</ul>"
    document.getElementById("id01").innerHTML = out;
}
Read a JSON File in Java

- JSON is also used to access data from a file
- A few libraries are available
- Example uses javax.json.*

```java
public class JsonRead {
    public static void main(String[] args) {
        Employee e = null;
        try {
            FileInputStream fileIn = new FileInputStream("employees.json");
            JsonReader reader = Json.createReader(fileIn);
            JsonArray employees = reader.readArray();
            JsonObject employee = employees.getJsonObject(0);
            JsonObject person = employee.getJsonObject("employee");
            System.out.println(person.getJsonString("firstName"));
            System.out.println(person);
            reader.close();
        } catch (IOException i) {
            i.printStackTrace();
            return;
        }
    }
}
```

Example

```
{ "employee": {
    "firstName": "Lokesh",
    "lastName": "Gupta",
    "website": "howtodoinjava.com"
} },
{ "employee": {
    "firstName": "Brian",
    "lastName": "Schultz",
    "website": "example.com"
} }
```

Library in javax.json.*
Did You Achieve the Lecture Objectives?

- Understand the need for serialization
- Understand various approaches to serialization
- Understand the use of JSON as a popular approach to serialization
- Understand how to access JSON data from JavaScript and Java