XML and JSON

Amos Azaria

XML Example

```
<University>
  <Student degree="PhD">
     <FirstName>Chaya</FirstName>
     <LastName>Glass</LastName>
     <id>111</id>
     <age>21</age>
     <Address>
        <Street>Hatamr 5</Street>
        <City>Ariel</City>
        <Zip>40792</Zip>
     </Address>
  </Student>
</University>
```

XML (eXtensible Markup Language)

- XML is used to store, share and communicate data. (Unlike HTML that is about displaying data)
- XML is wrapped information, not a programming language.
- Hierarchal data (tags must be properly nested).
- It is legal to have multiple elements with the same name (a list).
- Case-sensitive
- An XML that adheres to all the requirements is: "well formed".

XML Example (cont.)

```
<?xml version="1.0"?>
<University>
 <Student degree="PhD">
    <FirstName>Chaya</FirstName>
    <LastName>Glass</LastName>
    <id>111</id>
    <age>21</age>
    <Address>
       <Street>Hatamr 5</Street>
       <City>Ariel</City>
       <Zip>40792</Zip>
    </Address>
 </Student>
 <Student>
    <FirstName>Tal</FirstName>
    <LastName>Negev</LastName>
    <id>222</id>
    <Address>
       <Street>Rotem 53</Street>
       <City>Jerusalem</City>
       <Zip>54287</Zip>
    </Address>
 </Student>
 <Student/>
<!-- A comment about the file... -->
</University>
```

XML must contain only a single (root) element

degree here is an attribute (with a value of "PhD"). We usually use attributes for meta-data only (e.g. id), for easier fetching.

Closing tag for "Student"

This Student element doesn't have an "age" element

The "Address" element has 3 children elements (Street, City and Zip)

Empty element of Student.

Entity References

- What will happen if we want to use the '<' sign in an xml not as part of a tag?
- Suppose we have a tag of "failures" and a student has failed less than 3 times:

<failures> < 3 </failures>

<failures> < 3 </failures>

Is this a new element tag? < 3 …

Similar to the use of backslash in Java

<	<
>	>
&	&
'	'
"	"

Are all valid HTML documents also well formed XMLs?

- No! (HTML is out of the scope of this course)
- Examples (valid HTML invalid XML):
  - a paragraph (no close)
  - <input type="text" disabled /> (the disabled attribute doesn't have a value)
- Are all well formed XML documents also valid HTMLs?
  - No way!
- XHTML is HTML written as a well formed XML.

XML in JAVA

- **import** org.w3c.dom.*
- Create a DocumentBuilder
- Create a Document from a file (or InputStream)
- document.getDocumentElement()
- Repeat:
  - Node subNode = node.getElementsByTagName("Student").item(num);
  - .getAttribute("degree")
  - .getTextContent()

XML in JAVA (cont.)

- Code that parses previous XML to:
  - List<Student> studentList = new LinkedList<>();
  - **class** Student {**public** String **firstName**; **public** String **lastName**; **public int** id; **public int** age; **public** Address address;}
  - **class** Address {**public** String street; **public** String city; **public** String zip}

```
File inputFile = new File("student.xml");
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document doc = builder.parse(inputFile);
System.out.println("Root element :" + doc.getDocumentElement().getNodeName()); //Just print root (university)
NodeList nodeList = doc.getDocumentElement().getElementsByTagName("Student");
for (int studentIdx = 0; studentIdx < nodeList.getLength(); studentIdx++){
   Node studentNode = nodeList.item(studentIdx);
   if (studentNode.getNodeType() == Node.ELEMENT_NODE){
      Element element = (Element) studentNode;
      Student student = new Student();
      studentList.add(student);
      System.out.println("Degree : " + element.getAttribute("degree")); //just print degree ("PhD" when studentIdx=0)
      NodeList studentAllNodes = studentNode.getChildNodes();
      for (int stIdx = 0; stIdx < studentAllNodes.getLength(); stIdx++){
         Node stInnerNode = studentAllNodes.item(stIdx);
         switch (stInnerNode.getNodeName()){
            case "FirstName": student.firstName = stInnerNode.getTextContent(); break;
            case "LastName": student.lastName = stInnerNode.getTextContent(); break;
            case "id": student.id = Integer.parseInt(stInnerNode.getTextContent()); break;
            case "age": student.age = Integer.parseInt(stInnerNode.getTextContent()); break;
            case "Address":
               Address address = new Address();
               student.address = address;
               NodeList addressAllNodes = stInnerNode.getChildNodes();
               for (int adIdx = 0; adIdx < addressAllNodes.getLength(); adIdx++) {
                  Node adInnerNode = addressAllNodes.item(adIdx);
                  switch (adInnerNode.getNodeName()) {
                     case "Street": address.street = adInnerNode.getTextContent(); break;
                     case "City": address.city = adInnerNode.getTextContent(); break;
```

```
<?xml version="1.0"?>
<University>
  <Student degree="PhD">
    <FirstName>Chaya</FirstName>
    <LastName>Glass</LastName>
    <id>111</id>
    <age>21</age>
    <Address>
      <Street>Hatamr 5</Street>
      <City>Ariel</City>
      <Zip>40792</Zip>
    </Address>
  </Student>
  <Student>
    <FirstName>Tal</FirstName>
    <LastName>Negev</LastName>
    <id>222</id>
    <Address>
      <Street>Rotem 53</Street>
      <City>Jerusalem</City>
      <Zip>54287</Zip>
    </Address>
  </Student>
  <Student/>
<!-- A comment about the file… -->
</University>
```

- System.out.println(studentList.get(0).lastName)
  - Glass
- System.out.println(studentList.get(1).address.zip)
  - 54287
- System.out.println(studentList.get(2).id)
  - 0
- System.out.println(studentList.get(3).lastName)
  - IndexOutOfBoundsException…
- System.out.println(studentList.get(0).degree)
  - Won't even compile (we just printed the degree…)

XPATH

- XPATH allows us to point to a specific node/value in an XML.
- We don't need to traverse the whole tree, if we are only interested in a specific node.
- You can try using XPath (and XQuery) at: http://www.xpathtester.com/xpath

XPATH Example (Java)

- Suppose we would like to obtain the city of the second student's element.

One based (first item is Student[1] not Student[0])

XPATH basic syntax resembles the syntax used in file systems, but has many more operations.

- **University/Student[2]/Address/City**
- Usage example in Java:

```
XPathFactory xPathfactory = XPathFactory.newInstance();
XPath xpath = xPathfactory.newXPath();
XPathExpression expr =
xpath.compile("University/Student[2]/Address/City");
String city = (String)expr.evaluate(xmlDoc, XPathConstants.STRING);
```

University/Student[2]/Address/City

<City>Jerusalem</City>

Original XML

```
<?xml version="1.0"?>
<University>
 <Student degree="PhD">
    <FirstName>Chaya</FirstName>
    <LastName>Glass</LastName>
    <id>111</id>
    <age>21</age>
    <Address>
       <Street>Hatamr 5</Street>
       <City>Ariel</City>
       <Zip>40792</Zip>
    </Address>
 </Student>
 <Student>
    <FirstName>Tal</FirstName>
    <LastName>Negev</LastName>
    <id>222</id>
    <Address>
       <Street>Rotem 53</Street>
       <City>Jerusalem</City>
       <Zip>54287</Zip>
    </Address>
 </Student>
 <Student/>
<!-- A comment about the file... -->
</University>
```

More Operations

- Or |:
  - University/Student/FirstName | /University/Student/Address
- All descendants (recursive) //
  - University//Street
- All elements *
  - University/Student/*
- Parent path /..
- All ancestors /ancestor::*

Original XML

```
<?xml version="1.0"?>
<University>
 <Student degree="PhD">
   <FirstName>Chaya</FirstName>
   <LastName>Glass</LastName>
   <id>111</id>
   <age>21</age>
   <Address>
      <Street>Hatamr 5</Street>
      <City>Ariel</City>
      <Zip>40792</Zip>
   </Address>
 </Student>
 <Student>
   <FirstName>Tal</FirstName>
   <LastName>Negev</LastName>
   <id>222</id>
   <Address>
      <Street>Rotem 53</Street>
      <City>Jerusalem</City>
      <Zip>54287</Zip>
   </Address>
 </Student>
 <Student/>
<!-- A comment about the file... -->
</University>
```

University/Student/FirstName | /University/Student/Address

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
<FirstName>Chaya</FirstName>

<Address>
        <Street>Hatamr 5</Street>
        <City>Ariel</City>
        <Zip>40792</Zip>
    </Address>

<FirstName>Tal</FirstName>

<Address>
        <Street>Rotem 53</Street>
        <City>Jerusalem</City>
        <Zip>54287</Zip>
    </Address>

</result>
```

Original XML

```
<?xml version="1.0"?>
<University>
 <Student degree="PhD">
    <FirstName>Chaya</FirstName>
    <LastName>Glass</LastName>
    <id>111</id>
    <age>21</age>
    <Address>
       <Street>Hatamr 5</Street>
       <City>Ariel</City>
       <Zip>40792</Zip>
    </Address>
 </Student>
 <Student>
    <FirstName>Tal</FirstName>
    <LastName>Negev</LastName>
    <id>222</id>
    <Address>
       <Street>Rotem 53</Street>
       <City>Jerusalem</City>
       <Zip>54287</Zip>
    </Address>
 </Student>
 <Student/>
<!-- A comment about the file... -->
</University>
```

University//Street

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
<Street>Hatamr 5</Street>

<Street>Rotem 53</Street>

</result>
```

Original XML

```
<?xml version="1.0"?>
<University>
 <Student degree="PhD">
    <FirstName>Chaya</FirstName>
    <LastName>Glass</LastName>
    <id>111</id>
    <age>21</age>
    <Address>
       <Street>Hatamr 5</Street>
       <City>Ariel</City>
       <Zip>40792</Zip>
    </Address>
 </Student>
 <Student>
    <FirstName>Tal</FirstName>
    <LastName>Negev</LastName>
    <id>222</id>
    <Address>
       <Street>Rotem 53</Street>
       <City>Jerusalem</City>
       <Zip>54287</Zip>
    </Address>
 </Student>
 <Student/>
<!-- A comment about the file... -->
</University>
```

University/Student/*

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
<FirstName>Chaya</FirstName>
<LastName>Glass</LastName>
<id>111</id>
<age>21</age>
<Address>
        <Street>Hatamr 5</Street>
        <City>Ariel</City>
        <Zip>40792</Zip>
 </Address>
<FirstName>Tal</FirstName>
<LastName>Negev</LastName>
<id>222</id>
<Address>
        <Street>Rotem 53</Street>
        <City>Jerusalem</City>
        <Zip>54287</Zip>
    </Address>
</result>
```

Original XML

```
<?xml version="1.0"?>
<University>
 <Student degree="PhD">
    <FirstName>Chaya</FirstName>
    <LastName>Glass</LastName>
    <id>111</id>
    <age>21</age>
    <Address>
       <Street>Hatamr 5</Street>
       <City>Ariel</City>
       <Zip>40792</Zip>
    </Address>
 </Student>
 <Student>
    <FirstName>Tal</FirstName>
    <LastName>Negev</LastName>
    <id>222</id>
    <Address>
       <Street>Rotem 53</Street>
       <City>Jerusalem</City>
       <Zip>54287</Zip>
    </Address>
 </Student>
 <Student/>
<!-- A comment about the file... -->
</University>
```

XPath Selecting Attributes

- We add @ for selecting an attribute:
  - University/Student[1]/@degree
  - This just returns a string ("PhD"), not an XML!
- The following seems to return an XML (after adding an attribute to the second student):
  - University/Student/@degree

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
   PhD
   BSc
</result>
```

XPath conditions

- XPath supports adding conditions in brackets.
- Example:
  - doc("students.xml")/University/Student[age<25]

```
<Student degree="PhD">
    <FirstName>Chaya</FirstName>
    <LastName>Glass</LastName>
    <id>111</id>
    <age>21</age>
    <Address>
        <Street>Hatamr 5</Street>
        <City>Ariel</City>
        <Zip>40792</Zip>
    </Address>
 </Student>
```

Last Names of All Students Under 25

- We can continue our query also after a condition!
- /University/Student[age<25]/LastName
  - <LastName>Glass</LastName>

XQuery (XML Query)

- The equivalent of SQL for XMLs.
- All XPath queries are a valid XQuery query.
- FLWOR:
  - for (e.g. for $x in Xpath_expression) – similar to FROM
  - let (e.g. let $avg_age := avg(/University/Student/age))
    - similar to SET @variable
  - where (e.g. where $x/age < $avg_age)
  - order by (e.g. order by xs:int{$x/age})
  - return (e.g. return $x) – similar to SELECT

Without the cast, it will use lexicographical order (as age is a string)

XQuery Example

- for $x in /University/Student
  where $x/id > 0
  return $x

```
<?xml version="1.0" encoding="UTF-8"?>
<Student degree="PhD">
    <FirstName>Chaya</FirstName>
    <LastName>Glass</LastName>
    <id>111</id>
    <age>21</age>
    <Address>
        <Street>Hatamr 5</Street>
        <City>Ariel</City>
        <Zip>40792</Zip>
    </Address>
 </Student>
<Student>
    <FirstName>Tal</FirstName>
    <LastName>Negev</LastName>
    <id>222</id>
    <Address>
        <Street>Rotem 53</Street>
        <City>Jerusalem</City>
        <Zip>54287</Zip>
    </Address>
 </Student>
```

Original XML

```
<?xml version="1.0"?>
<University>
 <Student degree="PhD">
    <FirstName>Chaya</FirstName>
    <LastName>Glass</LastName>
    <id>111</id>
    <age>21</age>
    <Address>
        <Street>Hatamr 5</Street>
        <City>Ariel</City>
        <Zip>40792</Zip>
    </Address>
 </Student>
 <Student>
    <FirstName>Tal</FirstName>
    <LastName>Negev</LastName>
    <id>222</id>
    <Address>
        <Street>Rotem 53</Street>
        <City>Jerusalem</City>
        <Zip>54287</Zip>
    </Address>
 </Student>
 <Student/>
<!-- A comment about the file... -->
</University>
```

XQuery 2nd Example

```
for $x in /University/Student
    let $avg_age := avg(/University/Student/age)
    where $x/age < $avg_age + 1
    return $x/id
```

```
<?xml version="1.0" encoding="UTF-8"?>
<id>111</id>
```

Original XML

```
<?xml version="1.0"?>
<University>
 <Student degree="PhD">
    <FirstName>Chaya</FirstName>
    <LastName>Glass</LastName>
    <id>111</id>
    <age>21</age>
    <Address>
       <Street>Hatamr 5</Street>
       <City>Ariel</City>
       <Zip>40792</Zip>
    </Address>
 </Student>
 <Student>
    <FirstName>Tal</FirstName>
    <LastName>Negev</LastName>
    <id>222</id>
    <Address>
       <Street>Rotem 53</Street>
       <City>Jerusalem</City>
       <Zip>54287</Zip>
    </Address>
 </Student>
 <Student/>
<!-- A comment about the file... -->
</University>
```

Validation

- What if we would like to enforce that every student element has FirstName, LastName and id child elements?
- There are two document type definitions:
  - DTD (Document Type Definition)
  - XML Schema (XSD)

We will focus mostly on XML Schema, which is newer.

DTD example

The **square brackets [ ]** denote an internal subset (the declarations appear in the document itself)

Defines the root element of the document (University)

```
<!DOCTYPE University
[
<!ELEMENT student
(FirstName,LastName,Address*,id,age?)>
<!ELEMENT FirstName (#PCDATA)>
<!ELEMENT LastName (#PCDATA)>
]>
```

Element "student" must contain: FirstName,LastName, multiple Addresses ,id and optional age

FirstName and LastName are of type #PCDATA (parseable text data)

Note that DTD has no hierarchy.

XML Schema (XSD)

- An XML Schema (XSD) defines what the XML types can look like, and what is a "valid XML".
- XSD is itself a well-formed XML

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="University">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Student" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="FirstName" type="xs:string" />
              <xs:element name="LastName" type="xs:string" />
              <xs:element name="age" type="xs:int" />
              <xs:element name="Address">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="Street" type="xs:string" />
                    <xs:element name="City" type="xs:string" />
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          <xs:attribute name="degree" type="xs:string"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

This is where we get all the types from. Note the "xs" namespace prefix in all elements.

If we don't state either minOccurs or maxOccurs, the default is 1

"Address" is a complex type inside the "student" type

XSD supports many types

This allows the students to have a degree attribute (optional).
We could force the students to have a degree attribute by:
<xs:attribute name="degree" use="required"/>
And have a default attribute by:
<xs:attribute name="degree" default="BSc"/>

XML validated against an XSD

```
<?xml version="1.0"?>
<University>
  <Student degree="PhD">
     <FirstName>Chaya</FirstName>
     <LastName>Glass</LastName>
     <Address>
        <Street>Hatamr 5</Street>
        <City>Ariel</City>
        <Zip>40792</Zip>
     </Address>
     <age>21</age>
  </Student>
  <Student>
     <FirstName>Tom</FirstName>
     <LastName>Glow</LastName>
     <Address>
        <Street>Mishmar 5</Street>
        <City>Ariel</City>
     </Address>
  </Student>
</University>
```

Not valid.

1. Invalid content was found starting with element 'Address'. One of '{age}' is expected.
2. Invalid content was found starting with element 'Zip'. No child element is expected at this point.
3. Invalid content was found starting with element 'Address'. One of '{age}' is expected.

Correcting XML to match XSD

- We would usually correct the XML to match the XSD, but since we are learning XSD, we will do the opposite and make correct the XSD so that the given XML will match it.

Correcting XSD to match XML

- We will make the "age" element in the XSD optional by replacing it with:
  - <xs:element name="age" type="xs:int" minOccurs="0" />
- We will add an optional "zip" element in the XSD under Address (after Ci
  - <xs:element name="Zip" type="xs:int" minOccurs="0" />

Why don't we need to define maxOccurs?

Not valid.
1. Invalid content was found starting with element 'age'. No child element is expected at this point.

Correcting XSD to match XML

- We will define the complexType "student" to be a "xsd:all" rather than "xsd:sequence", this will allow any order over the elements of student.
  - So we replace "<xs:sequence>" with "<xs:all>" (and its matching tag).

XML is Valid!

Defining new types (e.g. non-empty string)

```
<xs:simpleType name="NonEmptyString">
    <xs:restriction base="xs:string">
        <xs:minLength value="1" />
        <xs:pattern value=".*[^\s].*" />
    </xs:restriction>
</xs:simpleType>
```

\s is whitespace.
^\s means not whitespace

Using it:

```
<xs:element name="lastName" type="NonEmptyString" />
```

JSON (JavaScript Object Notation)

JSON

- A skinny and compact data format.
- Designed for JavaScript, but widely used under other programming languages / platforms as well.
- {"idex": "value"}

JSON Example

```
{
 "University": {
  "Student": [
   {
    "FirstName": "Chaya",
    "LastName": "Glass",
    "Address": {
     "Street": "Hatamr 5",
     "City": "Ariel",
     "Zip": "40792"
    },
    "age": 21
   },
   {
    "FirstName": "Tom",
    "LastName": "Glow",
    "Address": {
     "Street": "Mishmar 5",
     "City": "Ariel"
    }
   }
  ]
 }
}
```

Array of "student"

Integer (no quotes)

Two additional types are booleans (true, false) and null.

JSON in JAVA

- There are several libraries for JSON in Java.
- The parsing itself is done similarly to the XML parsing.
- The following example parses our JSON input to a List<Student> studentList, using the json.org library:
  - compile group: 'org.json', name: 'json', version: '20140107' in Gradle dependencies
  - or download the jar from http://central.maven.org/maven2/org/json/json/20140107/json-20140107.jar and add it to your 'lib' directory (just as you have done with connector/J)

```
String jsonTxt = new String(Files.readAllBytes(Paths.get("students.json")));
JSONObject json = new JSONObject(jsonTxt);
JSONArray jsonStudentArray = json.getJSONObject("University").getJSONArray("Student");
for (int studentIdx = 0; studentIdx < jsonStudentArray.length(); studentIdx++){
   JSONObject currentStudent = jsonStudentArray.getJSONObject(studentIdx);
   Student student = new Student();
   studentList.add(student);
   JSONArray studentInner = currentStudent.names(); //array of keys only!
   for (int stInnerIdx = 0; stInnerIdx < studentInner.length(); stInnerIdx++){
      String currentKey = studentInner.getString(stInnerIdx);
      switch (currentKey){
         case "FirstName": student.firstName = currentStudent.getString(currentKey); break;
         case "LastName": student.lastName = currentStudent.getString(currentKey); break;
         case "id": student.id = currentStudent.getInt(currentKey); break;
         case "age": student.age = currentStudent.getInt(currentKey); break;
         case "Address":
            Address address = new Address();
            student.address = address;
            JSONObject addressObject = currentStudent.getJSONObject(currentKey);
            if (addressObject.has("Street"))
               address.street = addressObject.getString("Street");
            if (addressObject.has("City"))
               address.city = addressObject.getString("City");
            if (addressObject.has("Zip"))
               address.zip = addressObject.getString("Zip");
            break;
```

JSON Example

```
{
 "University": {
  "Student": [
   {
    "FirstName": "Chaya",
    "LastName": "Glass",
    "Address": {
     "Street": "Hatamr 5",
     "City": "Ariel",
     "Zip": "40792"
    },
    "age": 21
   },
   {
    "FirstName": "Tom",
    "LastName": "Glow",
    "Address": {
     "Street": "Mishmar 5",
     "City": "Ariel"
    }
   }
  ]
 }
}
```

- System.out.println(studentList.get(0).firstName)
  - Chaya
- System.out.println(studentList.get(0).address.zip)
  - 40792
- System.out.println(studentList.get(0).age)
  - 21
- System.out.println(studentList.get(1).address.street)
  - Mishmar 5

Note that if we call, for example:
currentStudent.getString("fdsafd")
we get:
org.json.JSONException: JSONObject["fdsafd"] not found.

JSON Schema

- Exists, but:
  - Not standardized.
  - Not widely used.

JSONPath

- Similar to XPath, but not well developed (newer), not standardized, not widely in use.