Lecture 21

Files

Streams

Readers and writers

XML and JSON data in Java

SAX and DOM parsers

IT TALENTS
Training Camp

Files

- In Java all files(including directories) are objects
- Files(including directories) are considered part of the I/O API
- The File class is java.io.File
- The constructor creates a File only by its path
  - Path is the way to the file
  - Relative path – the path from the root of the project to the accessed file
  - Absolute path – path from the root of the file system tree to the accessed file

IT TALENTS
Training Camp

Files

```
File file1 = new File("C:\\ITTalents\test.txt");
File file2 = new File("..","\\test.txt");
File file3 = new File("C:\\ITTalents");
System.out.println(file1==file2);
boolean file1Exists = file1.exists();
boolean file1IsFile = file1.isFile();
boolean file3IsDir = file3.isDirectory();
File files[]= file3.listFiles();
file1.delete();
```

File constructors

Using absolute path to file

Using relative path to file

Checks the file for existance

Prints false, although the file is one and the same

Checks file type- Either file or directory

Lists all files in the directory

Deletes the file

Exercise

- Create a class FileDemo with a main method
  - Check for folder *iotest* in directory D:\javaTest and if it does not exist, create it firstly
  - Create a new file *test.txt* in *iotest*
  - Via Windows Explorer, create 3 files in the iotest dir
  - List all of the files from dir istest and write their names in the console
  - Delete files, with names, starting with letter „t“

Introduction

- Input/Output – basic and main mechanism for reading and writing data from and to external resources

  for example reading from files, writing to files

- System.out.println() is already known approach for writing data

  therefore System.out.println() is part of I/O

- I/O functionality in java is kept in the package java.io

Basic Input and Output

Input and Output is a flow of data i.e of bytes and because of this flow I/O objects are called streams

Example: Imagine live streaming of TV program or Radio streaming

Logically *Input* is used for reading (i.e the data comes in) and *Output* for writing (i.e the data goes out)

IT TALENTS
Training Camp

Basic Input and Output

The streaming means and derives from the following features

- The data in the stream is ordered – we can't swap the position of the bytes
- The data flows sequentially – i.e byte X can't be read before all bytes from 1 to X-1 have been read. Data can't be accessed randomly.
- Reading and Writing always starts from the beginning
- Reading can proceed till end of file

Basic Input and Output

- Reading and writing occurs only in one direction – forwards. Streams can't go backwards – i.e once a byte has been read it can't be read again by the same stream
- When reading or writing to file a connection to the file is open so after reading or writing the stream MUST be closed
- Each connection should be closed every time the work with the file ends
- End of file is presented through byte value of -1

Streams in Java

- Byte streams
- Character streams
- Buffered streams
- Object streams
- Scanner and PrintStream

Streams in Java

- On top of all streams are
  - InputStream for reading
  - OutputStream for writing
- InputStream and OutputStream are abstract and can't be instantiated

Streams in Java

- InputStream reads data through the method read()
- read(byte[] array) – writes the read data into the byte array
- OutputStream writes data through the method write()
- If the read byte is -1 the End Of File is reached
- OutputStream writes the data into the file
- IOException and FileNotFoundException should be caught and handled

IT TALENTS
Training Camp

File Streams

- FileStreams are basic byte streams
- Using the methods of InputStream and OutputStream:
  - Read the bytes from test.txt
  - Write some new data into it

IT TALENTS
Training Camp

FileInput/OutputStream

```
FileInputStream input = new FileInputStream(file);
int b=0;
while(b!=-1){
    b= input.read();
}
input.close();
```

While the read byte is different from EOF

Reading byte and assigning it to b

Closing the stream

```
FileOutputStream output = new FileOutputStream(file);
output.write(24);
output.write('c');
output.close();
```

Writing bytes

Closing the stream

IT TALENTS Training Camp

Exercise

Create a program that compare two .jpg files

IT TALENTS
Training Camp

Character Streams

- The character streams are Reader and Writer
- Reader and Writer are specified only for characters
- Reader and Writer are abstact and refer to charachter files as InputStream and OutputStream do for byte streams
- FileWriter and FileReader can be used to write/read text to/from files.

Scanner and PrintStream

- Scanner is a utility class which uses regular expressions for easier parsing a character source
- Scanner is not a stream but can work as such

  or it can use a stream to file as well
- PrintStream can be used for writing into a file

Scanner and PrintStream

Defining Scanner through a file

```
Scanner sc = new Scanner(file);
while(sc.hasNextLine()) {
    System.out.println(sc.nextLine());
}
```

```
InputStream stream = new FileInputStream(file);
Scanner sc = new Scanner(stream);
while(sc.hasNextLine()) {
    System.out.println(sc.nextLine());
}
```

Defining Scanner through a stream

Introduction to XML

- *e***X***tensible* **M***arkup* **L***anguage* (XML)
- An XML element is made up of a start tag, an end tag and data in between
- Example:
  - *<director> Krasi Stoev </director>*
- Example of another element with same value:
  - <actor> Krasi Stoev </actor>
- XML tags are case-sensitive:
  - <CITY> <City> <city>
- XML can abbreviate empty elements, for example:
- <married></married> can be abbreviated to </married>

Introducion to XML

- Attributes:
  - An attribute is a name-value pair separated by an equal sign (=)
  - Example
    - *<City ZIP="66493"> Sofia </City>*
  - Attributes are used to attack additional secondary information to an element.

Introduction to XML

- A basic XML document is an XML element that can, but might not, include nested XML elements.

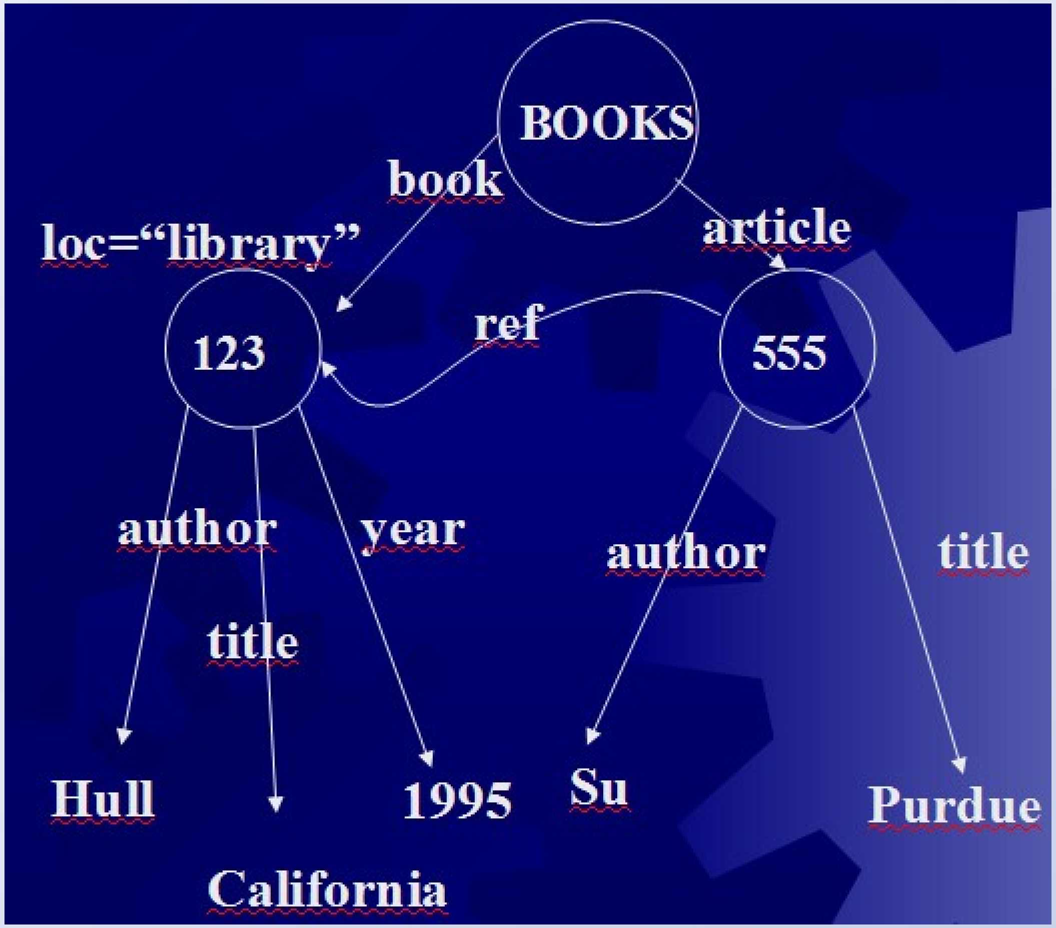
- Example:

```
<books>
      <book isbn="123">
            <title> Second Chance </title>
            <author> Matthew Dunn </author>
      </book>
 </books>
```

IT TALENTS
Training Camp

XML Data Model Example

```
<BOOKS>
  <book id="123" loc="library">
    <author>Hull</author>
    <title>California</title>
    <year> 1995 </year>
  </book>
  <article id="555" ref="123">
    <author>Su</author>
    <title> Purdue</title>
  </article>
</BOOKS>
```

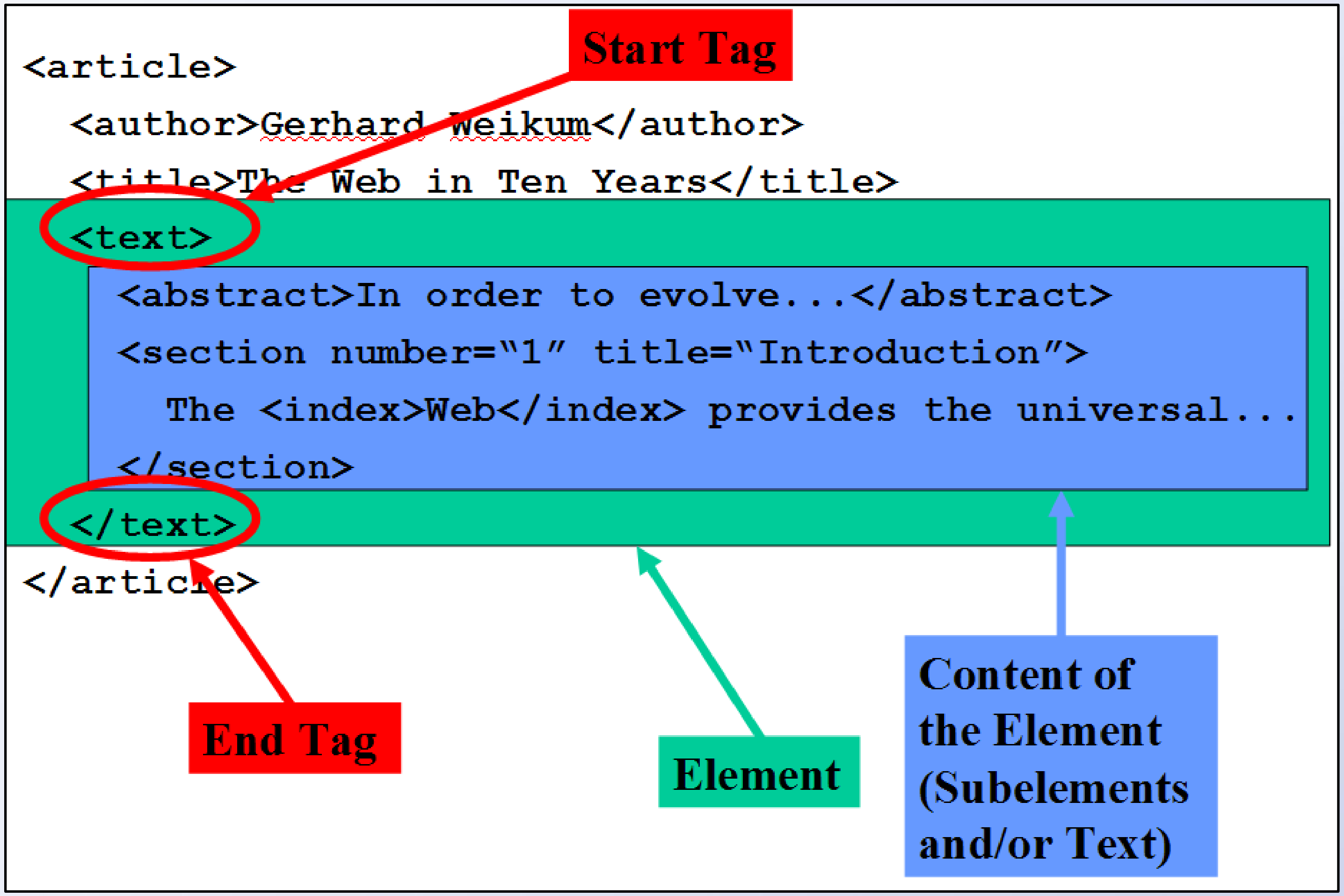


XML Rules and Regulations

- All elements must have an end tag
- All elements must be cleanly nested (overlapping elements are not allowed)
- All attribute values must be enclosed in quotation marks
- Each document must have a unique first element, the root node.

IT TALENTS
Training Camp

A Simple XML Document



```
<article>
  <author>Gerhard Weikum</author>
  <title>The Web in Ten Years</title>
  <text>
    <abstract>In order to evolve...</abstract>
    <section number="1" title="Introduction">
      The <index>Web</index> provides the universal...
    </section>
  </text>
</article>
```

More about XML on http://www.tutorialspoint.com/xml/index.htm

XML Parsers in Java

- What is XML Parsing?
  - Parsing XML refers to going through XML document to access data or to modify data in one or other way.
  - XML Parser provides way how to access or modify data present in an XML document. Java provides multiple options to parse XML document. Following are various types of parsers which are commonly used to parse XML documents.

IT TALENTS
Training Camp

XML Parsers

- **Dom Parser** - Parses the document by loading the complete contents of the document and creating its complete hiearchical tree in memory.
- **SAX Parser** - Parses the document on event based triggers. Does not load the complete document into the memory.
- **JDOM Parser** - Parses the document in similar fashion to DOM parser but in more easier way.
- **StAX Parser** - Parses the document in similar fashion to SAX parser but in more efficient way.
- **XPath Parser** - Parses the XML based on expression and is used extensively in conjuction with XSLT.
- **DOM4J Parser** - A java library to parse XML, XPath and XSLT using Java Collections Framework , provides support for DOM, SAX and JAXP.

IT TALENTS
Training Camp

DOM Parsers

- When to use?
  - You need to know a lot about the structure of a document
  - You need to move parts of the document around (you might want to sort certain elements, for example)
  - You need to use the information in the document more than once
- What you get ?
  - you get back a tree structure that contains all of the elements of your document. The DOM provides a variety of functions you can use to examine the contents and structure of the document.

SAX Parsers

- When to use?
  - You can process the XML document in a linear fashion from the top down
  - The document is not deeply nested
  - You are processing a very large XML document whose DOM tree would consume too much memory. Typical DOM implementations use ten bytes of memory to represent one byte of XML
  - The problem to be solved involves only part of the XML document
  - Data is available as soon as it is seen by the parser, so SAX works well for an XML document that arrives over a stream

IT TALENTS
Training Camp

Parsing XML in Java

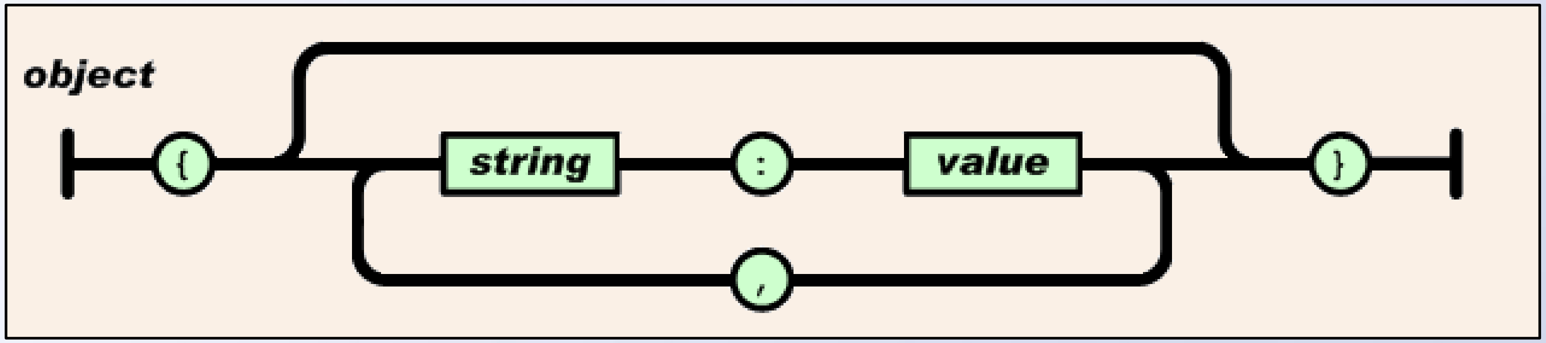
- http://xstream.codehaus.org/tutorial.html
- For homework

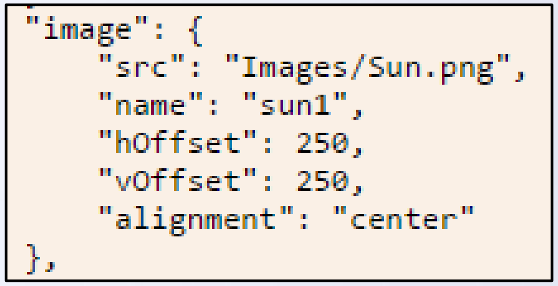
Introduction to JSON

- **J**ava **S**cript **O**bject **N**otation – **JSON**
- It is easy for humans to read and write. It is easy for machines to parse and generate.
- Built on two structures:
  - A collection of name/value pairs.
  - An ordered list of values.

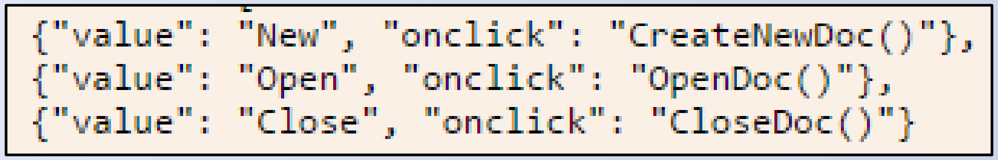
Introduction to JSON

- Object – an unordered set of name/value pairs. Starts with **{** and ends with **}**. Each name is followed by **:** and each pair is separated by **,**





```
"image": {
    "src": "Images/Sun.png",
    "name": "sun1",
    "hOffset": 250,
    "vOffset": 250,
    "alignment": "center"
},
```

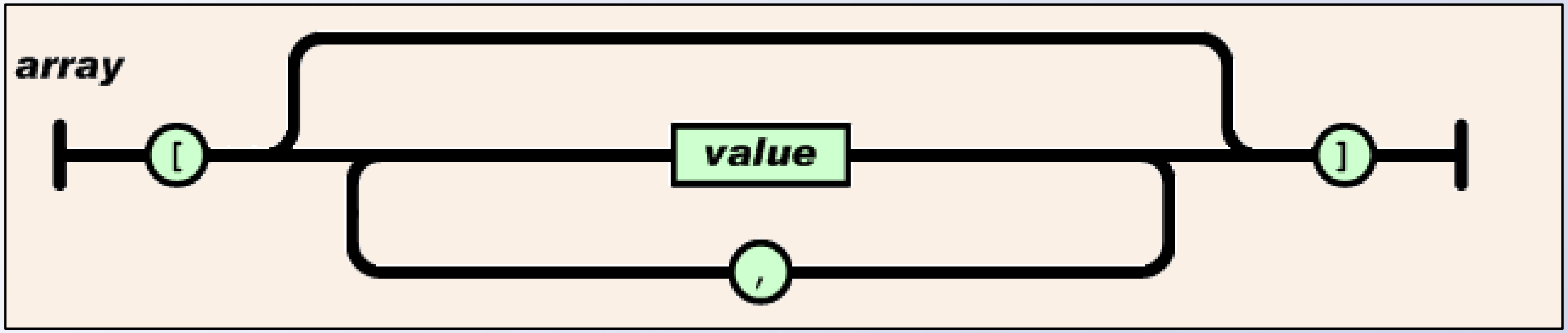


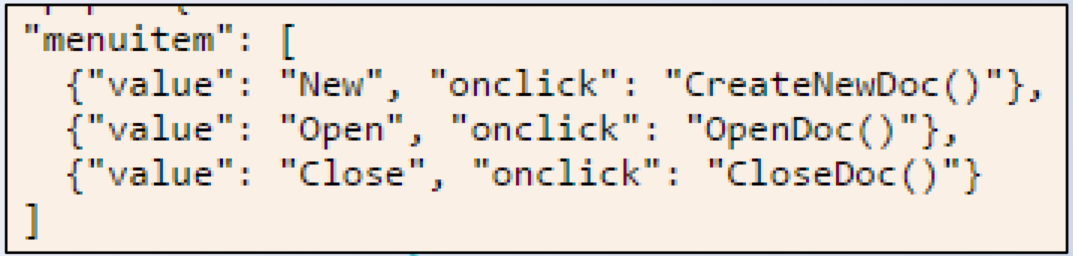
```
{"value": "New", "onclick": "CreateNewDoc()"},
{"value": "Open", "onclick": "OpenDoc()"},
{"value": "Close", "onclick": "CloseDoc()"}
```

IT TALENTS
Training Camp

Introduction to JSON

- Array – an ordered collection of values. Begins with **[** and ends with **]**. Values are separated by **,**

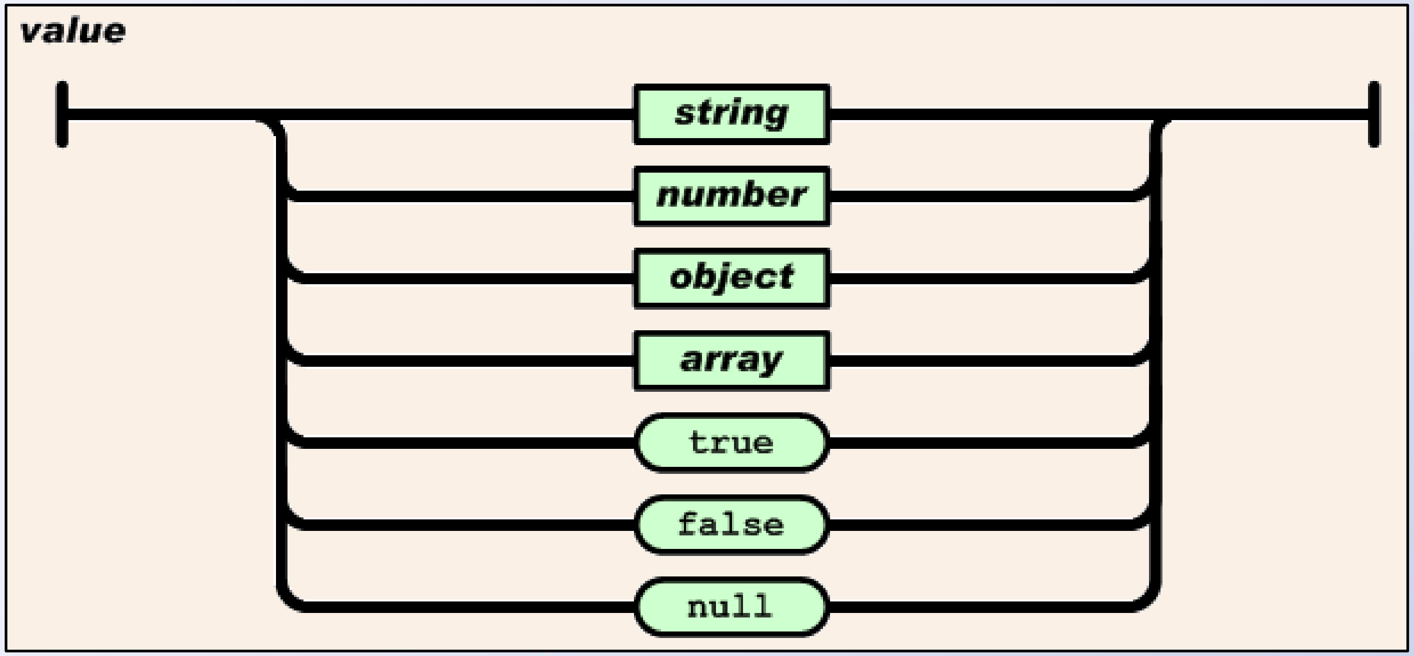




```
"menuitem": [
  {"value": "New", "onclick": "CreateNewDoc()"},
  {"value": "Open", "onclick": "OpenDoc()"},
  {"value": "Close", "onclick": "CloseDoc()"}
]
```

Introduction to JSON

- Value - can be a string in double quotes, or a number, or true or false or null, or an object or an array. These structures can be nested.



IT TALENTS
Training Camp

Parsing JSON in Java

- http://www.tutorialspoint.com/json/json_java_example.htm
- For homework