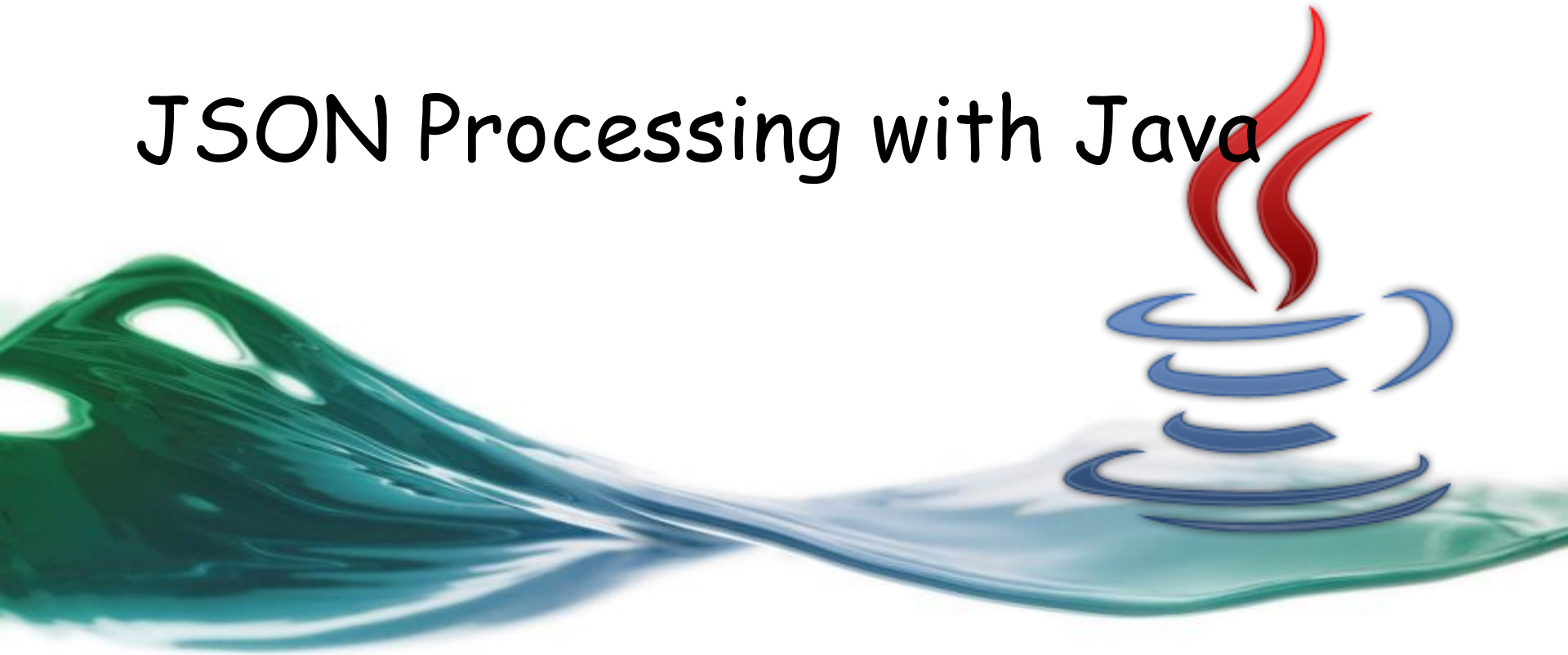


Java Programming Course

JSON Processing with Java



Faculty of Information Technologies
Industrial University of Ho Chi Minh City

Session objectives

JSON Introduction

JSON structure

Java API for JSON Processing



JSON Introduction

<http://www.json.org/>

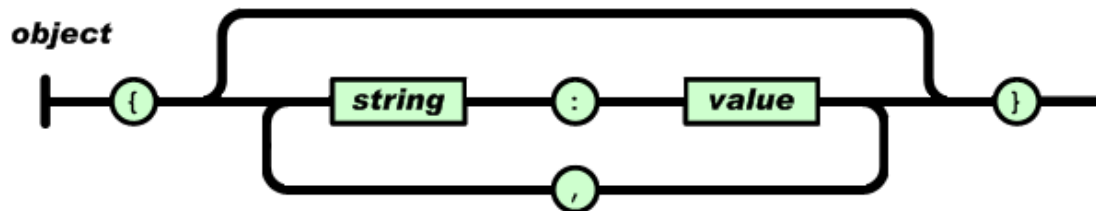
- **JSON** (JavaScript Object Notation) is a lightweight data-interchange format.
 - It is easy for humans to read and write.
 - It is easy for machines to parse and generate.
 - It is based on a subset of the [JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999](#).
 - JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.
- JSON is often used in Ajax applications, configurations, databases, and RESTful web services. All popular websites offer JSON as the data exchange format with their RESTful web services.

JSON structure (1)

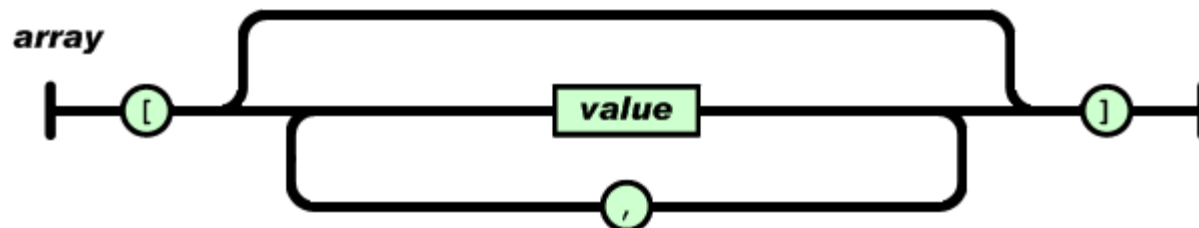
- JSON is built on two structures:
 - A collection of name/value pairs. In various languages, this is realized as an *object*, record, struct, dictionary, hash table, keyed list, or associative array.
 - An ordered list of values. In most languages, this is realized as an *array*, vector, list, or sequence.

JSON structure (2)

- In JSON, they take on these forms:
 - An **object** is an unordered set of name/value pairs. An object begins with { (left brace) and ends with } (right brace). Each name is followed by : (colon) and the name/value pairs are separated by , (comma).

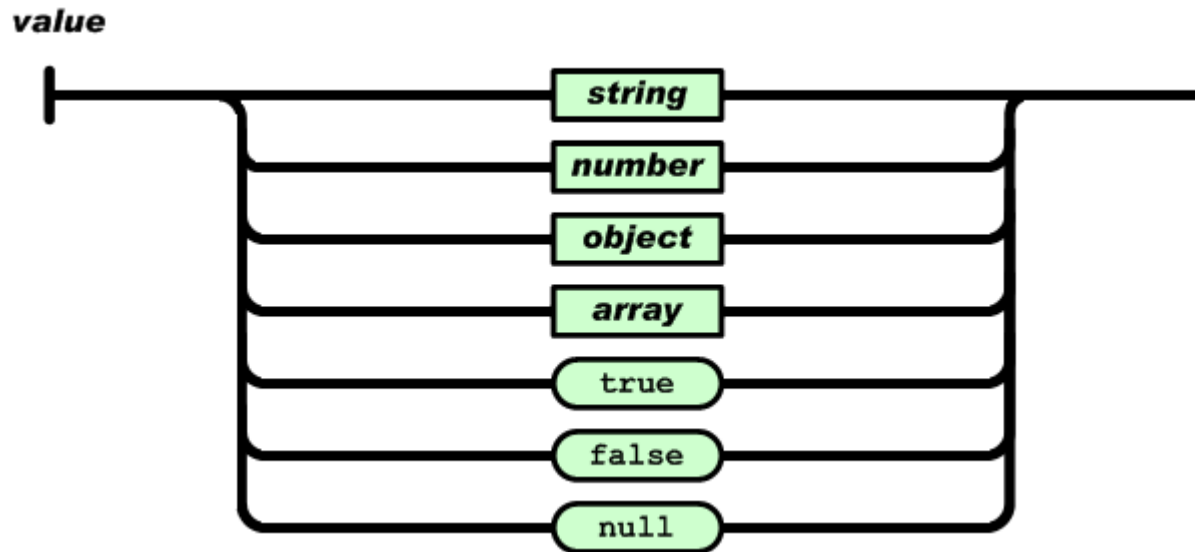


- An **array** is an ordered collection of values. An array begins with [(left bracket) and ends with] (right bracket). Values are separated by , (comma).



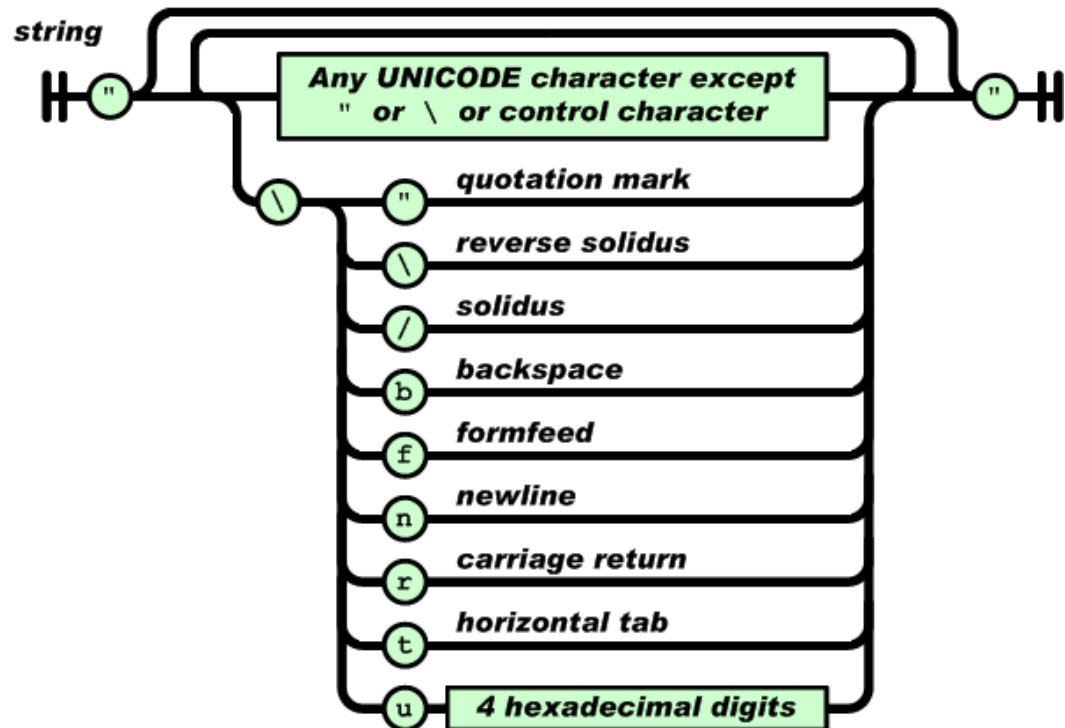
JSON structure (3)

- A **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.



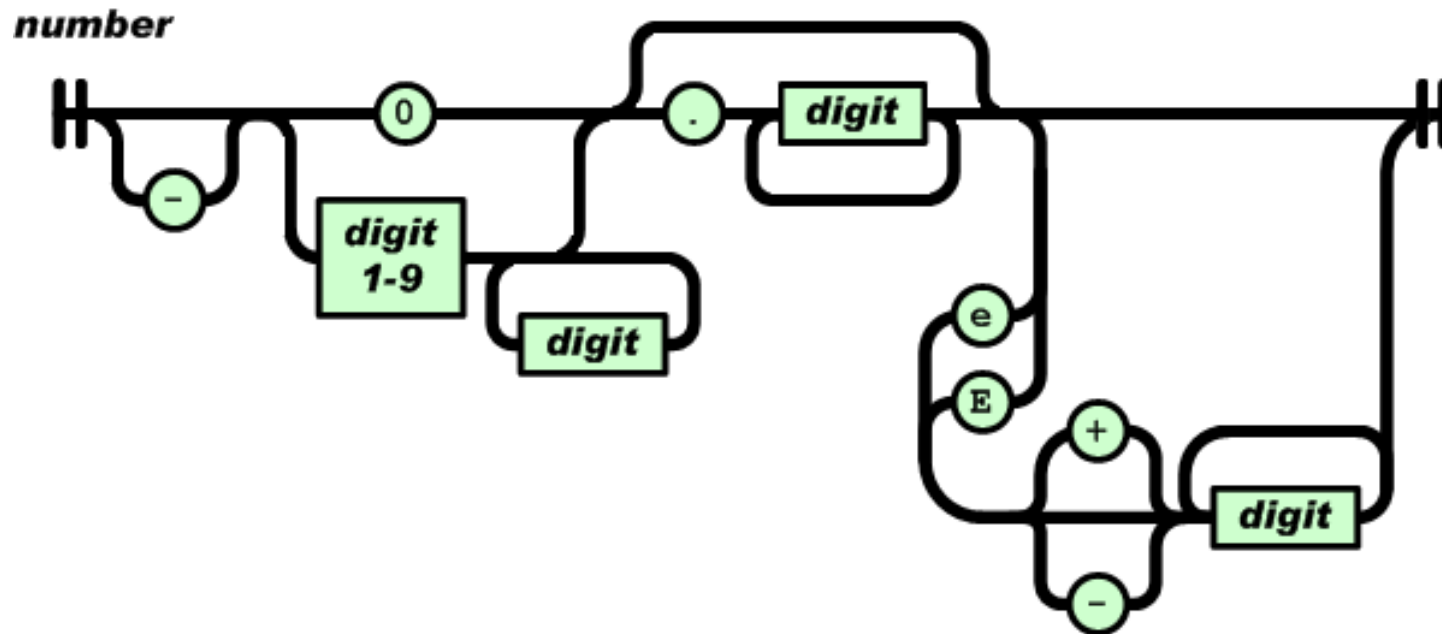
JSON structure (4)

- A *string* is a sequence of zero or more Unicode characters, wrapped in double quotes, using backslash escapes. A character is represented as a single character string. A string is very much like a C or Java string.



JSON structure (5)

- A **number** is very much like a C or Java number, except that the octal and hexadecimal formats are not used.



Sample json document & rule

```
{ } cust.json ⌵
1 {
2   "firstName": "John",
3   "lastName": "Smith",
4   "age": 25,
5   "address": {
6     "streetAddress": "21 2nd Street",
7     "city": "New York",
8     "state": "NY",
9     "postalCode": 10021
10  },
11  "phoneNumbers": [
12    {
13      "type": "home",
14      "number": "212 555-1234"
15    },
16    {
17      "type": "fax",
18      "number": "646 555-4567"
19    }
20  ]
21 }
```

```
object
  {}
  { members }
members
  pair
  pair , members
pair
  string : value
array
  []
  [ elements ]
elements
  value
  value , elements
value
  string
  number
  object
  array
  true
  false
  null
```

Java API for JSON Processing

- The Java API for JSON Processing ([JSR 353](#)) provides portable APIs to parse, generate, transform, and query JSON using object model and streaming APIs.
- It produces and consumes JSON text in a streaming fashion (similar to StAX API for XML) and allows to build a Java object model for JSON text using API classes (similar to DOM API for XML).

JSON Processing - The Object Model API (1)

- The Object Model API
 - The object model API creates a random-access, tree-like structure that represents the JSON data in memory. The tree can then be navigated and queried.
 - This programming model is the most flexible and enables processing that requires random access to the complete contents of the tree. However, it is often not as efficient as the streaming model and requires more memory.
 - The object model API is similar to the Document Object Model (DOM) API for XML.
 - It is a high-level API that provides immutable object models for JSON object and array structures. These JSON structures are represented as object models using the Java types `JsonObject` and `JsonArray`.

JSON Processing - The Object Model API (2)

- The main classes and interfaces in the object model API

Class or Interface	Description
Json	Contains static methods to create JSON readers, writers, builders, and their factory objects.
JsonGenerator	Writes JSON data to a stream one value at a time.
JsonReader	Reads JSON data from a stream and creates an object model in memory.
JsonObjectBuilder	Create an object model or an array model in memory by adding values from application code.
JsonArrayBuilder	
JsonWriter	Writes an object model from memory to a stream.
JsonValue	Represent data types for values in JSON data.
JsonObject	
JsonArray	
JsonString	
JsonNumber	

JSON Processing - The Object Model API (3)

- `JsonObject`, `JsonArray`, `JsonString`, and `JsonNumber` are subtypes of `JsonValue`. These are constants defined in the API for null, true, and false JSON values.
- The object model API uses builder patterns to create these object models from scratch. Application code can use the interface `JsonObjectBuilder` to create models that represent JSON objects. The resulting model is of type `JsonObject`. Application code can use the interface `JsonArrayBuilder` to create models that represent JSON arrays. The resulting model is of type `JsonArray`.
- These object models can also be created from an input source (such as *InputStream* or *Reader*) using the interface `JsonReader`. Similarly, these object models can be written to an output source (such as *OutputStream* or *Writer*) using the class `JsonWriter`.

Mapping between JSON and Java entities

JSON	Java
string	java.lang.String
number	java.lang.Number
true false	java.lang.Boolean
null	null
array	java.util.List
object	java.util.Map

On decoding:

The default concrete class of *java.util.List* is *org.json.simple.JSONArray*

The default concrete class of *java.util.Map* is *org.json.simple.JSONObject*.

Encoding JSON in Java

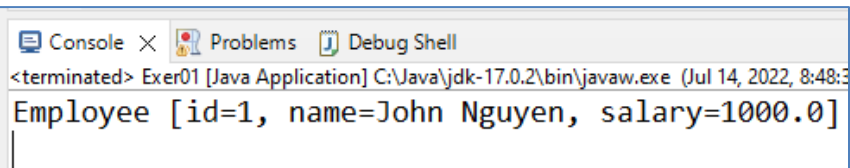
```
public static void main(String[] args) {  
    JsonObjectBuilder jsonBuilder = Json.createObjectBuilder();  
    Employee employee = new Employee(11, "John Nguyen", 1000);  
    JsonObject jo = jsonBuilder  
        .add("id", employee.getId())  
        .add("name", employee.getName())  
        .add("salary", employee.getSalary())  
        .build();  
    System.out.println(jo.toString());  
}
```

```
{  
    "id": 1,  
    "name": "John Nguyen",  
    "salary": 1000.0  
}
```

```
<dependencies>  
    <!-- https://mvnrepository.com/artifact/javax.json/javax.json-api -->  
    <dependency>  
        <groupId>javax.json</groupId>  
        <artifactId>json-api</artifactId>  
        <version>1.1.4</version>  
    </dependency>  
    <!-- https://mvnrepository.com/artifact/org.glassfish/javax.json -->  
    <dependency>  
        <groupId>org.glassfish</groupId>  
        <artifactId>javax.json</artifactId>  
        <version>1.1.4</version>  
    </dependency>  
</dependencies>
```

Decoding JSON in Java

```
public static void main(String[] args) {  
  
    String json = "{\"id\":1,\"name\":\"John Nguyen\",\"salary\":1000.0}";  
    JsonReader reader = Json.createReader(new StringReader(json));  
    JsonObject jo = reader.readObject();  
  
    Employee employee = new Employee(jo.getJsonNumber("id").longValue(),  
                                     jo.getString("name"),  
                                     jo.getJsonNumber("salary").doubleValue());  
    System.out.println(employee);  
}
```



The screenshot shows an IDE window with three tabs: 'Console', 'Problems', and 'Debug Shell'. The 'Console' tab is active and displays the output of the Java application. The text in the console is: '<terminated> Exer01 [Java Application] C:\Java\jdk-17.0.2\bin\javaw.exe (Jul 14, 2022, 8:48:3)' followed by 'Employee [id=1, name=John Nguyen, salary=1000.0]' on the next line. A cursor is visible at the end of the output line.

JSON Processing - The Streaming API (1)

- The Streaming API
 - The streaming API provides a way to parse and generate JSON in a streaming fashion.
 - It hands over parsing and generation control to the programmer.
 - The streaming API provides an event-based parser and allows an application developer to ask for the next event rather than handling the event in a callback. This gives a developer more procedural control over the JSON processing. Application code can process or discard the parser event and ask for the next event (pull the event).

JSON Processing - The Streaming API (2)

- The streaming API is similar to the Streaming API for XML (StAX) and consists of the interfaces `JsonParser` and `JsonGenerator`.
- `JsonParser` contains methods to parse JSON data using the streaming model.
- `JsonGenerator` contains methods to write JSON data to an output source. Table 2 lists the main classes and interfaces in the streaming API.

JSON Processing - The Streaming API (3)

- The main classes and interfaces in the streaming API

Class or Interface	Description
Json	Contains static methods to create JSON parsers, generators, and their factory objects.
JsonParser	Represents an event-based parser that can read JSON data from a stream.
JsonGenerator	Writes JSON data to a stream one value at a time

JSON Processing - The Streaming API (3)

- `JsonParser` provides forward, read-only access to JSON data using the pull parsing programming model. In this model, the application code controls the thread and calls methods in the parser interface to move the parser forward or to obtain JSON data from the current state of the parser.
- `JsonGenerator` provides methods to write JSON data to a stream. The generator can be used to write name/value pairs in JSON objects and values in JSON arrays.
- The streaming API is a low-level API designed to process large amounts of JSON data efficiently. Other JSON frameworks (such as *JSON binding*) can be implemented using this API.

Decoding JSON in Java - Stream API

```
public static void main(String[] args) {  
    // Parse back  
    final String result = "{\"name\":\"Falco\", \"age\":3, \"bitable\":false}";  
    final JsonParser parser = Json.createParser(  
new StringReader(result));  
    String key = null;  
    String value = null;  
    while (parser.hasNext()) {  
        final Event event = parser.next();  
        switch (event) {  
            case KEY_NAME:  
                key = parser.getString();  
                System.out.println(key);  
                break;  
            case VALUE_STRING:  
                value = parser.getString();  
                System.out.println(value);  
                break;  
        }  
    }  
    parser.close();  
}
```



Conclusion

- The Java API for JSON Processing provides the following capabilities:
 - Parsing input streams into immutable objects or event streams
 - Writing event streams or immutable objects to output streams
 - Programmatically navigating immutable objects
 - Programmatically building immutable objects with builders

FAQ





That's all for this session!

Thank you all for your attention and patient !