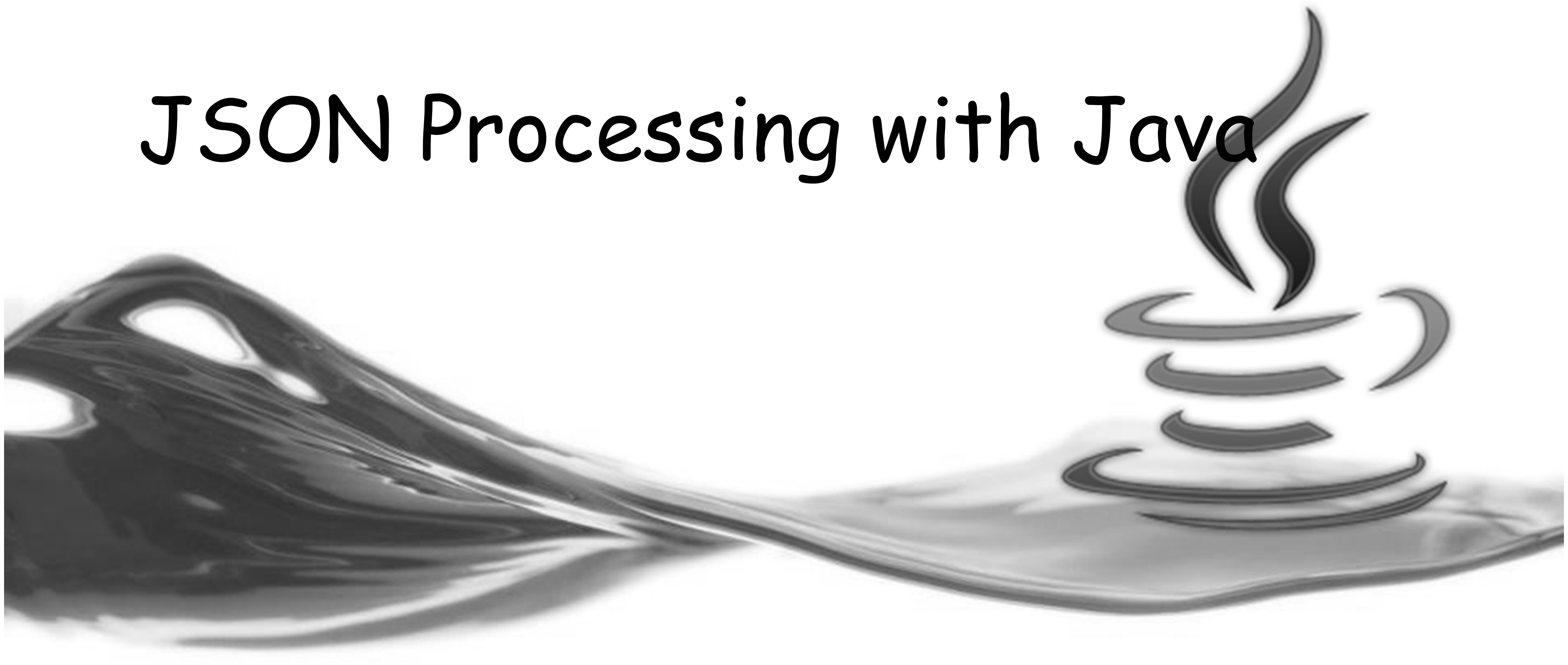
Java Programming Course

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/

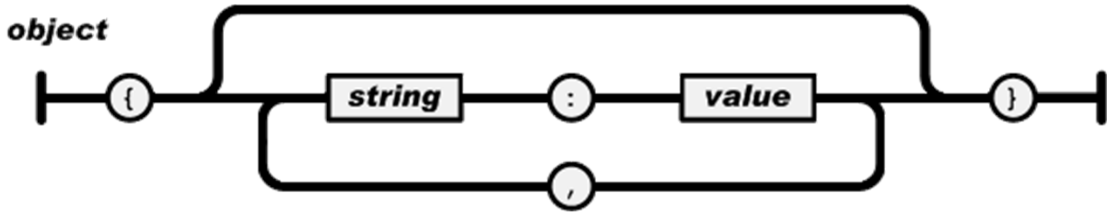
“

- **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 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.

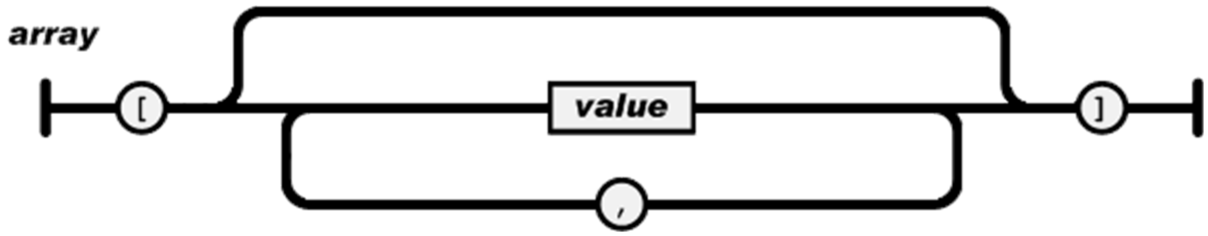
”

JSON structure (1)

- 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).

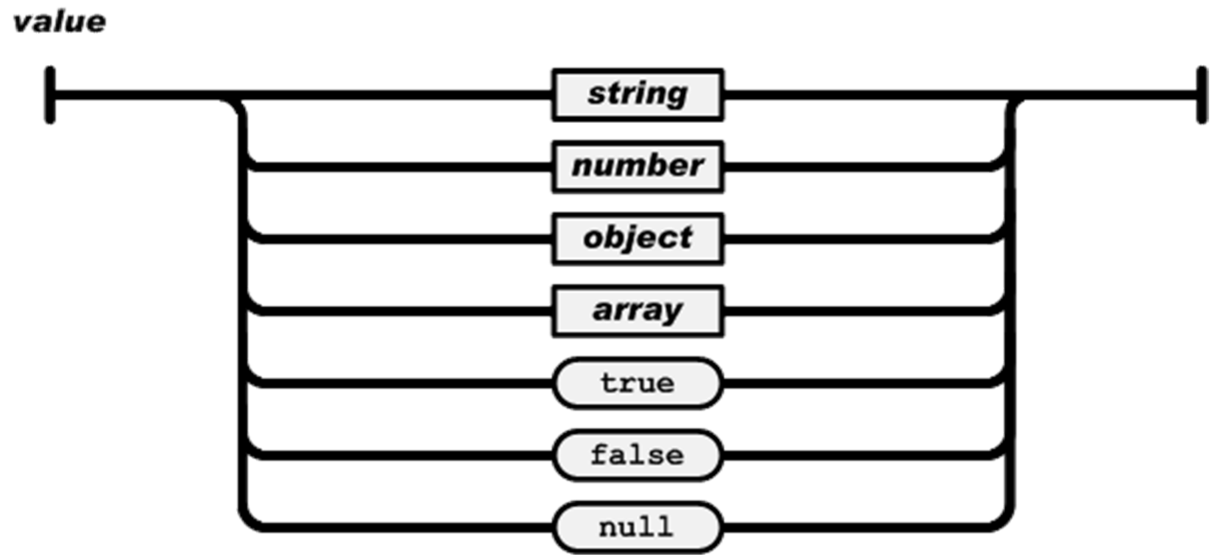


  - An *array* is an ordered collection of values. An array begins with [(left bracket) and ends with] (right bracket). Values are separated by , (comma).



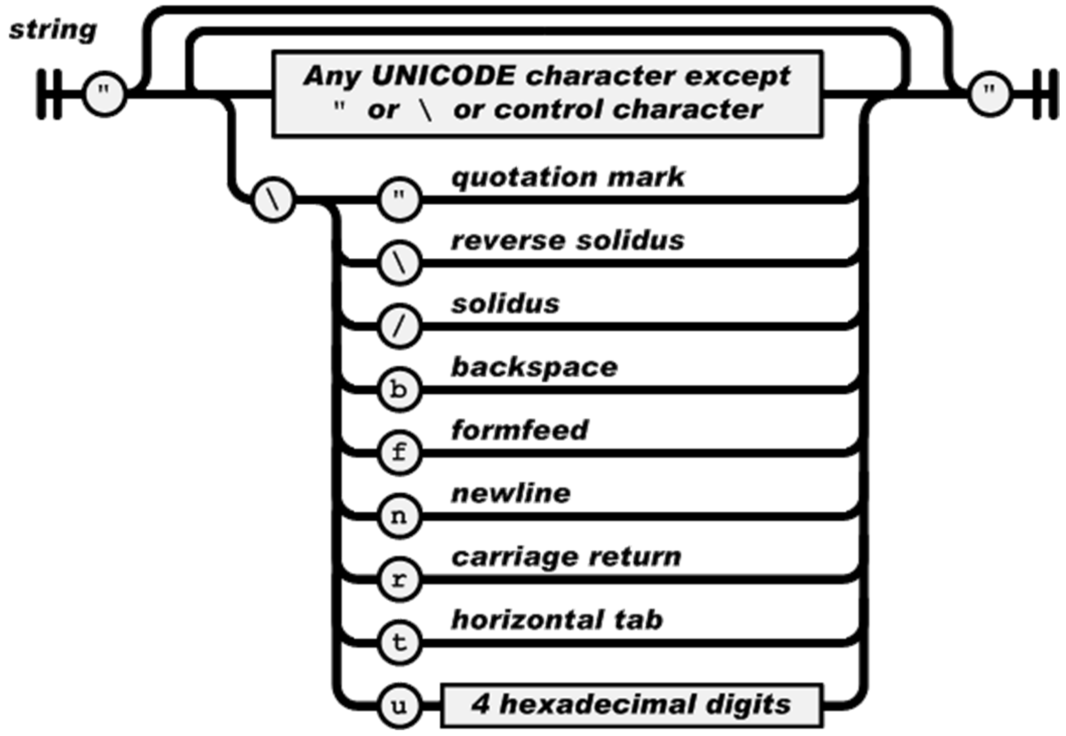
JSON structure (2)

- *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.



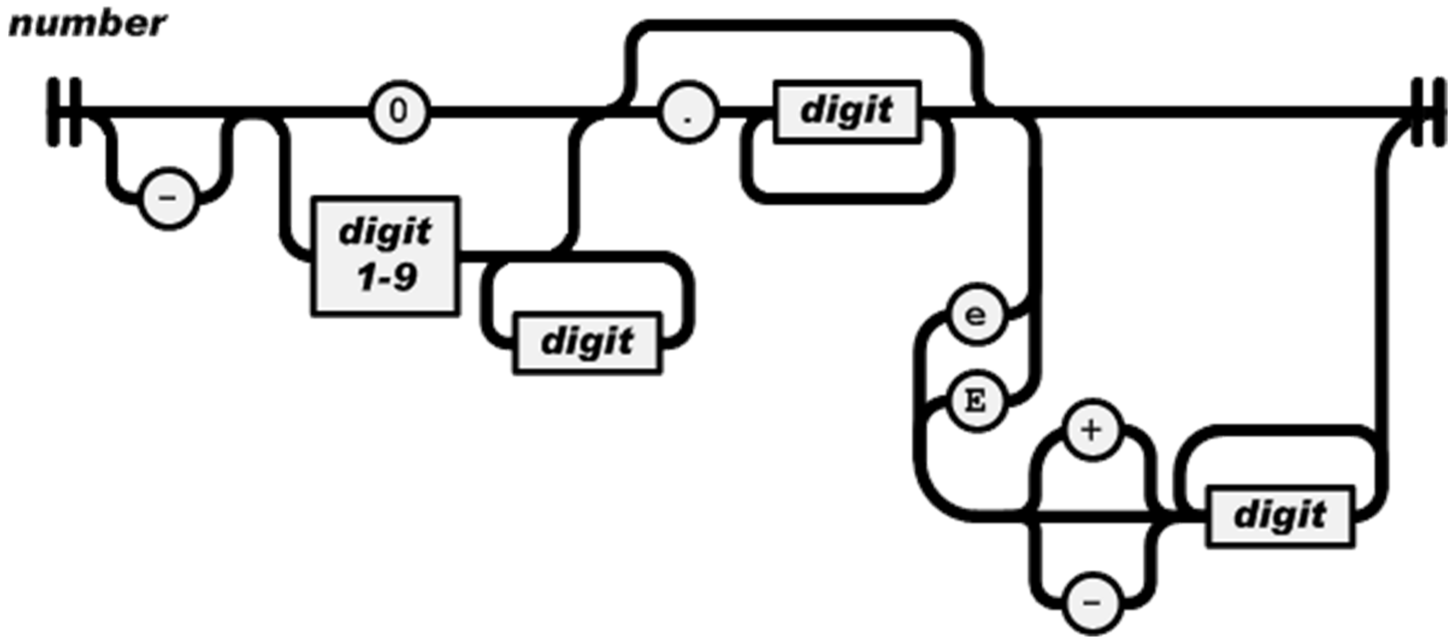
JSON structure (3)

- *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.

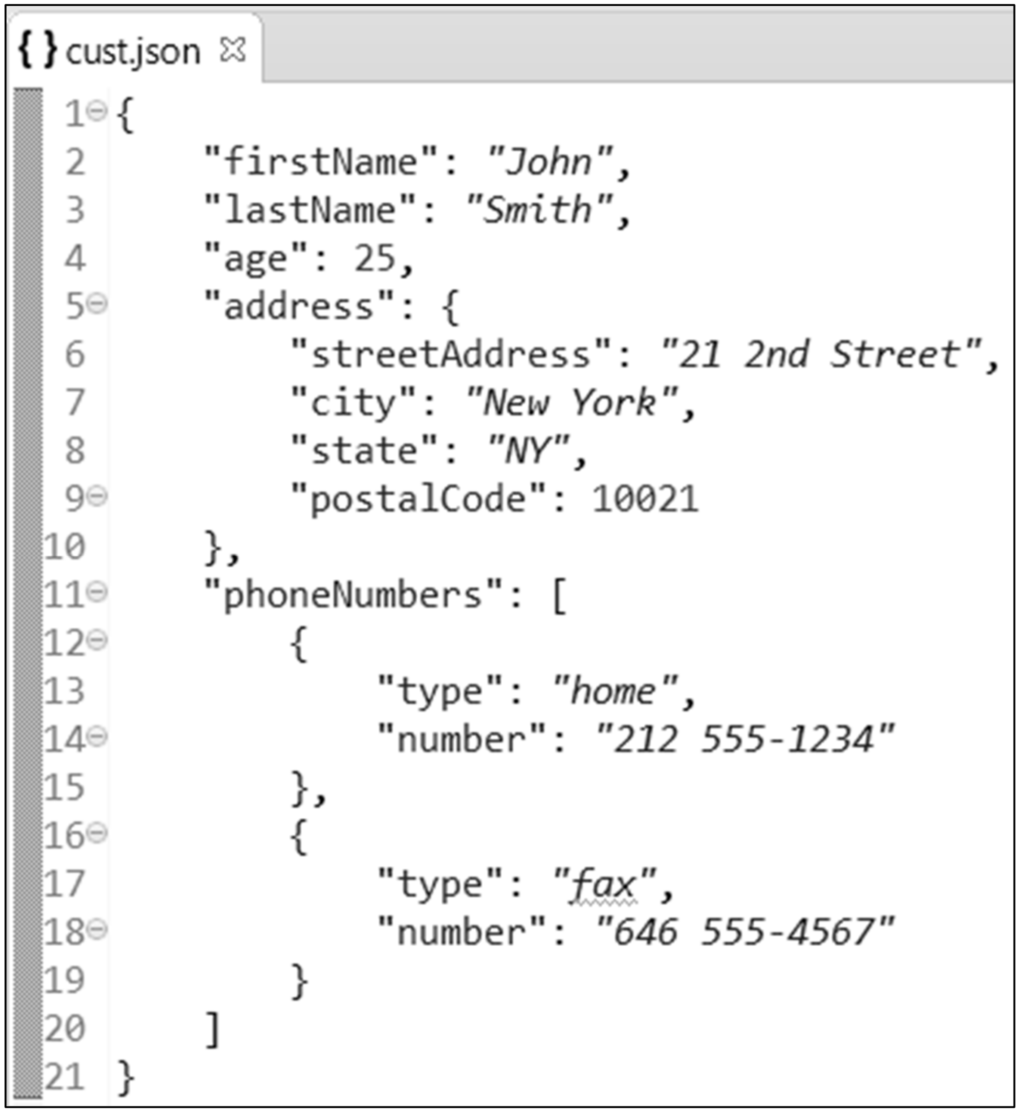


JSON structure (4)

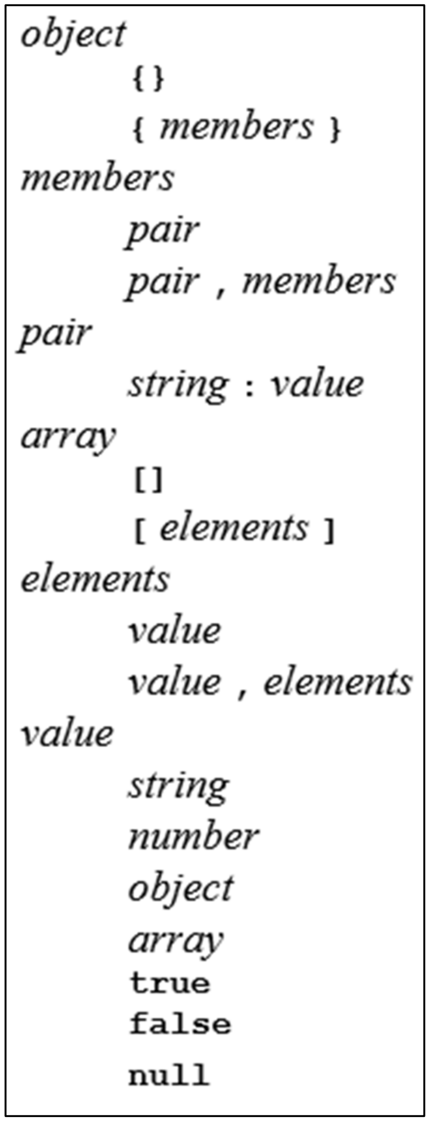
- *A number* is very much like a *C* or Java number, except that the octal and hexadecimal formats are not used.



Sample json document & rule


{ } cust.json

```
{
    "firstName": "John",
    "lastName": "Smith",
    "age": 25,
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": 10021
    },
    "phoneNumbers": [
        {
            "type": "home",
            "number": "212 555-1234"
        },
        {
            "type": "fax",
            "number": "646 555-4567"
        }
    ]
}
```

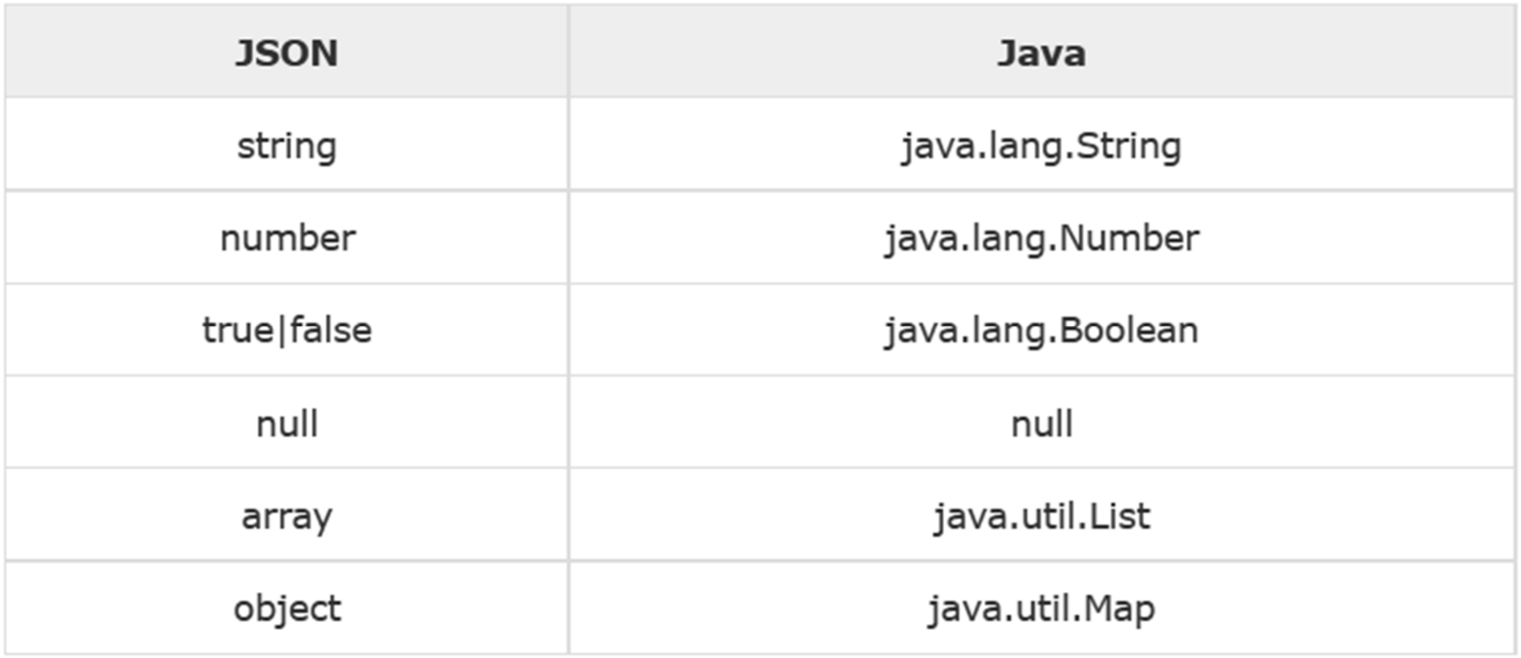


```
object
        {}
        { members }
members
        pair
        pair , members
pair
        string : value
array
        []
        [ elements ]
elements
        value
        value , elements
value
        string
        number
        object
        array
        true
        false
        null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

Java API for JSON Processing

- JSR 374 Specification
- JSON Processing (JSON-P) is a Java API to process (for e.g. parse, generate, transform and query) JSON messages.
- 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).

Mapping between JSON and Java entities



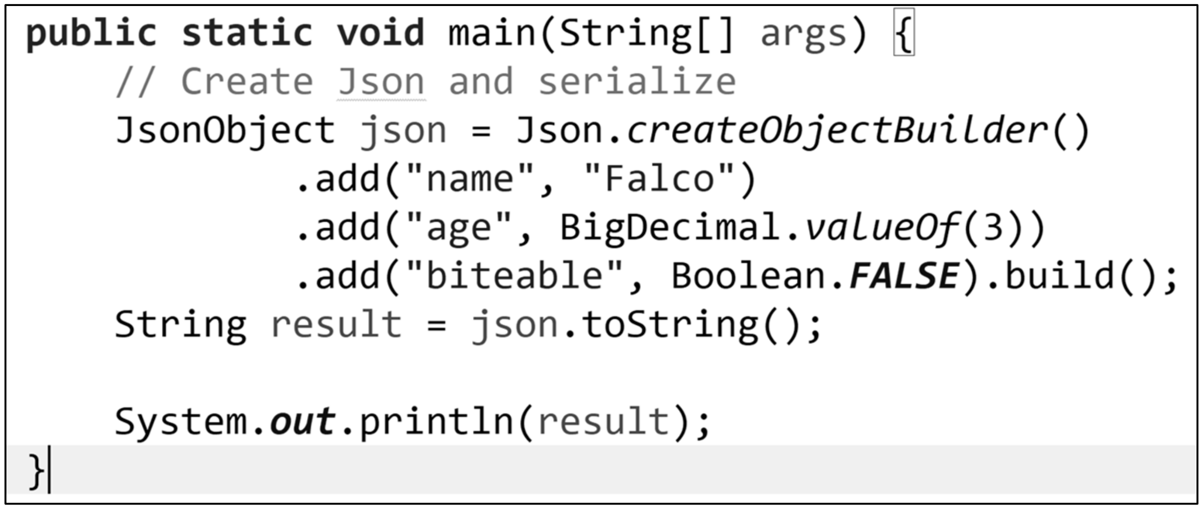
JSON	Java
string	java.lang.String
number	java.lang.Number
true|false	java.lang.Boolean
null	null
array	java.util.List
object	java.util.Map

On decoding:

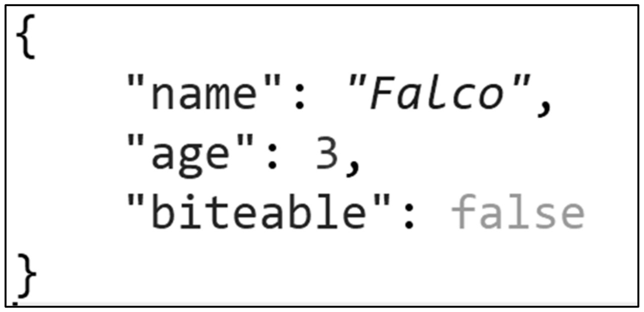
The default concrete class of *java.util.List* is *org.json.simple.JSONArray*

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

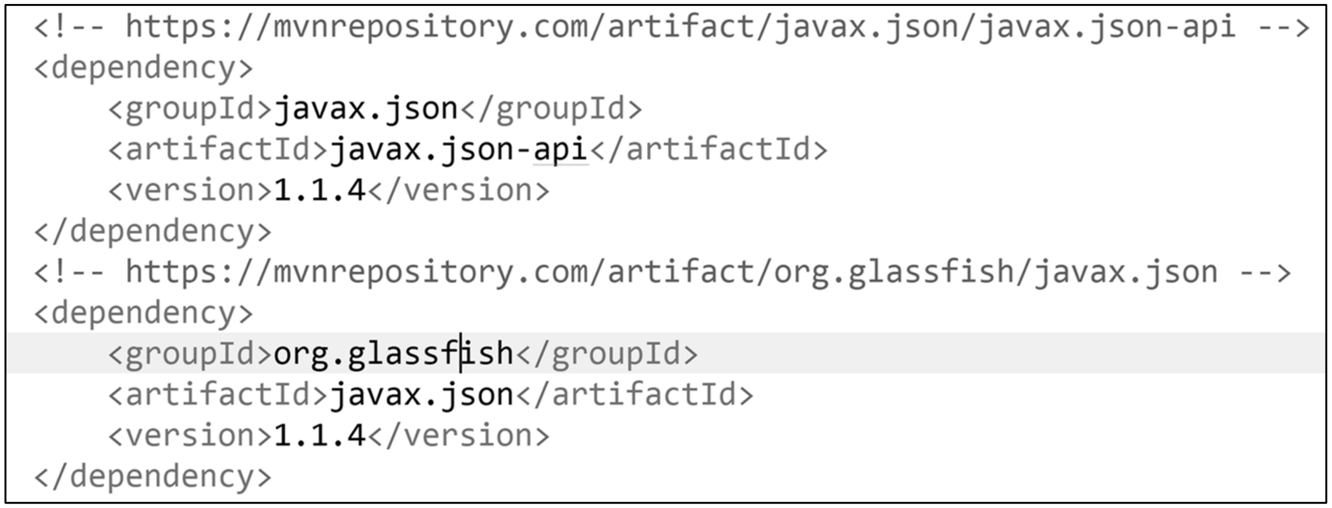
Encoding JSON in Java



```java
public static void main(String[] args) {
    // Create Json and serialize
    JsonObject json = Json.createObjectBuilder()
            .add("name", "Falco")
            .add("age", BigDecimal.valueOf(3))
            .add("biteable", Boolean.FALSE).build();
    String result = json.toString();

    System.out.println(result);
}
```



```json
{
    "name": "Falco",
    "age": 3,
    "biteable": false
}
```

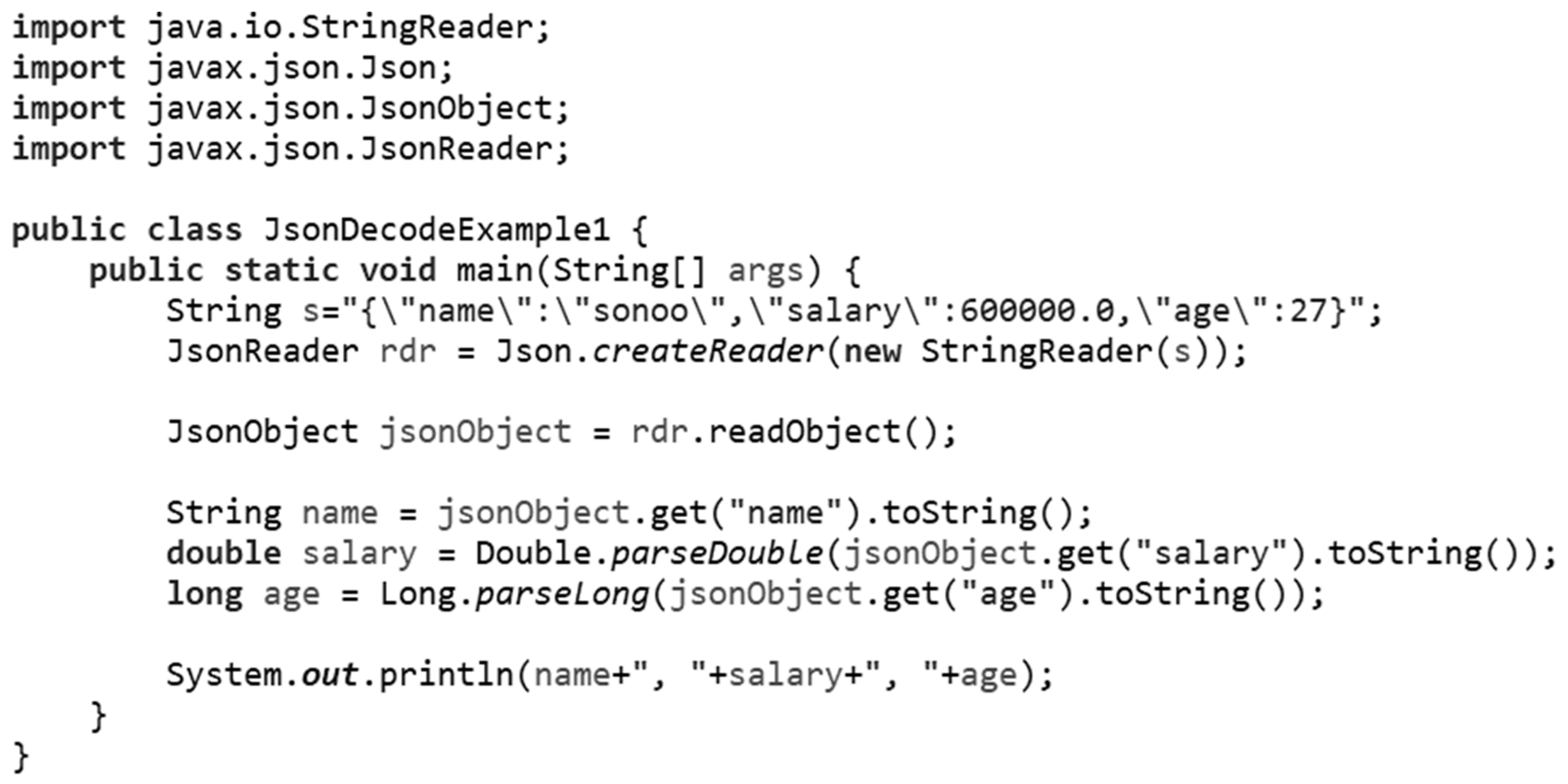


```xml
<!-- https://mvnrepository.com/artifact/javax.json/javax.json-api -->
<dependency>
    <groupId>javax.json</groupId>
    <artifactId>javax.json-api</artifactId>
    <version>1.1.4</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.glassfish/javax.json -->
<dependency>
    <groupId>org.glassfish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

Encoding JSON in Java

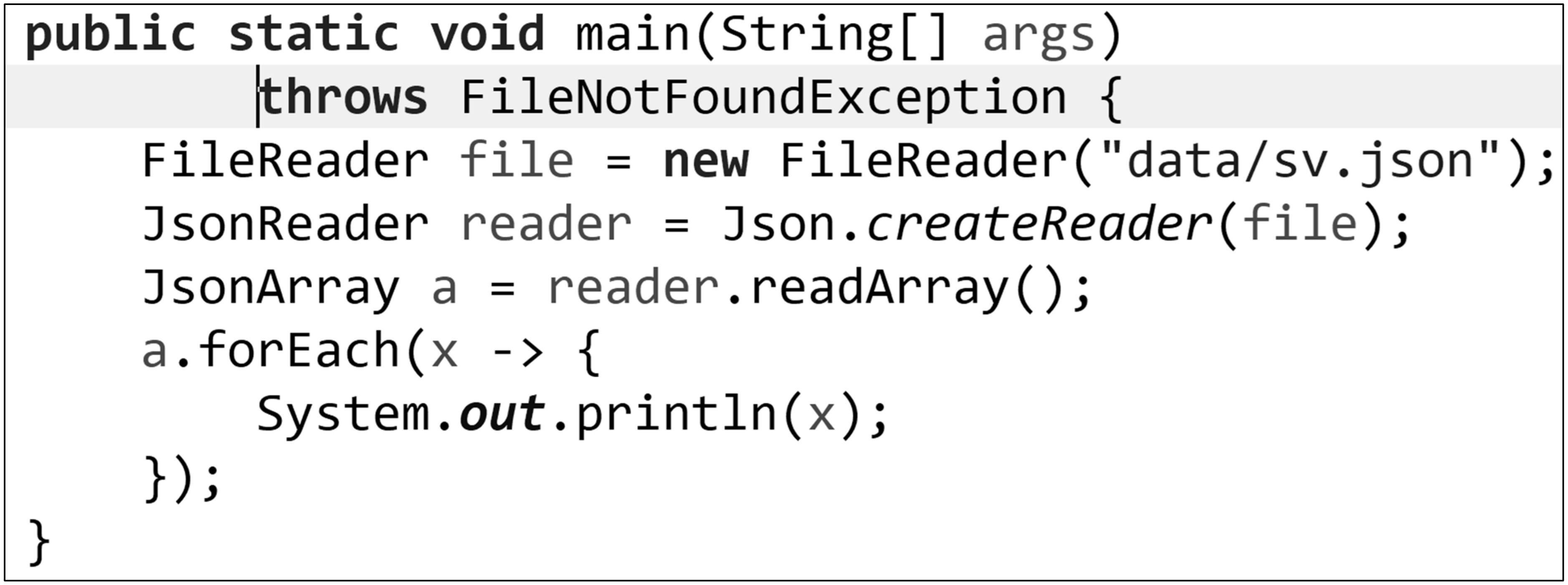
```java
public static void main(String[] args) {
    JsonObjectBuilder objectBuilder = Json.createObjectBuilder();

    JsonObject x1 = objectBuilder
            .add("mssv", "111")
            .add("hoten", "Binh")
            .build();
    JsonObject x2 = objectBuilder
            .add("mssv", "112")
            .add("hoten", "Hoa")
            .build();

    JsonArrayBuilder arrayBuilder = Json.createArrayBuilder();
    JsonArray x = arrayBuilder.add(x1).add(x2).build();

    System.out.println(x);
}
```

Decoding JSON in Java



```java
import java.io.StringReader;
import javax.json.Json;
import javax.json.JsonObject;
import javax.json.JsonReader;

public class JsonDecodeExample1 {
    public static void main(String[] args) {
        String s="{\"name\":\"sonoo\",\"salary\":600000.0,\"age\":27}";
        JsonReader rdr = Json.createReader(new StringReader(s));

        JsonObject jsonObject = rdr.readObject();

        String name = jsonObject.get("name").toString();
        double salary = Double.parseDouble(jsonObject.get("salary").toString());
        long age = Long.parseLong(jsonObject.get("age").toString());

        System.out.println(name+", "+salary+", "+age);
    }
}
```

Decoding JSON in Java



```java
public static void main(String[] args)
        throws FileNotFoundException {
    FileReader file = new FileReader("data/sv.json");
    JsonReader reader = Json.createReader(file);
    JsonArray a = reader.readArray();
    a.forEach(x -> {
        System.out.println(x);
    });
}
```

Decoding JSON in Java – Stream API

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

The working of GSON

- GSON is an Java library to serialize and deserialize Java objects to (and from) JSON.
- It provides two methods :
  - Gson.toJson to serialize java objects.
  - Gson.fromJson to deserialize json objects.
- GSON Example
  - Serialization:

```
Gson gson = new Gson();
Employee employee = new Employee(1, "Anna", 100000);
String json = gson.toJson(employee);
System.out.println(json)
```

The working of GSON

- GSON Example
  - Deserialization:

```
Gson gson = new Gson();
String x =
"{\"id\":1,\"name\":\"Anna\",\"salary\":100000.0}";
Employee e = gson.fromJson(x, Employee.class);
System.out.println(e.getSalary());
```

FAQ

That's all for this session!

Thank you all for your attention and patient !