# Java Basic for Tester

*Reading and Writing XML, JSON in Java*

## Agenda

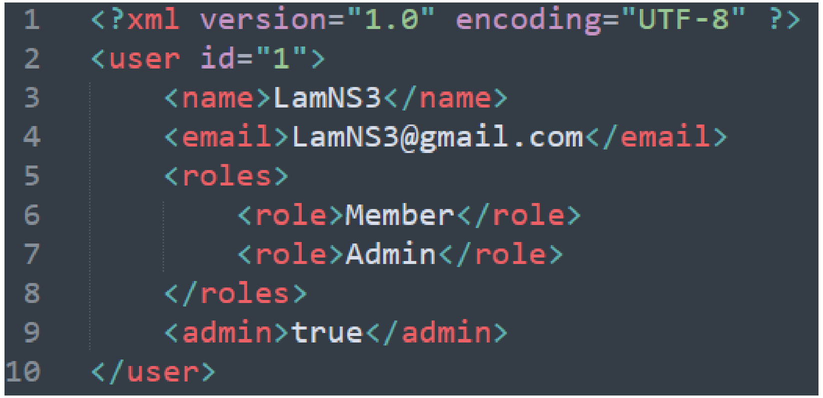
- *Reading and Writing XML in Java*
- *Reading and Writing JSON in Java*

## Reading and Writing XML in Java

XML — short for eXtensible Markup Language — is a popular format for exchanging data between web services, computers, and front-ends after JSON. It was defined by W3C in 1998 and has a markup structure similar to HTML. Despite having a markup like HTML, XML is commonly used for storing and transporting data.

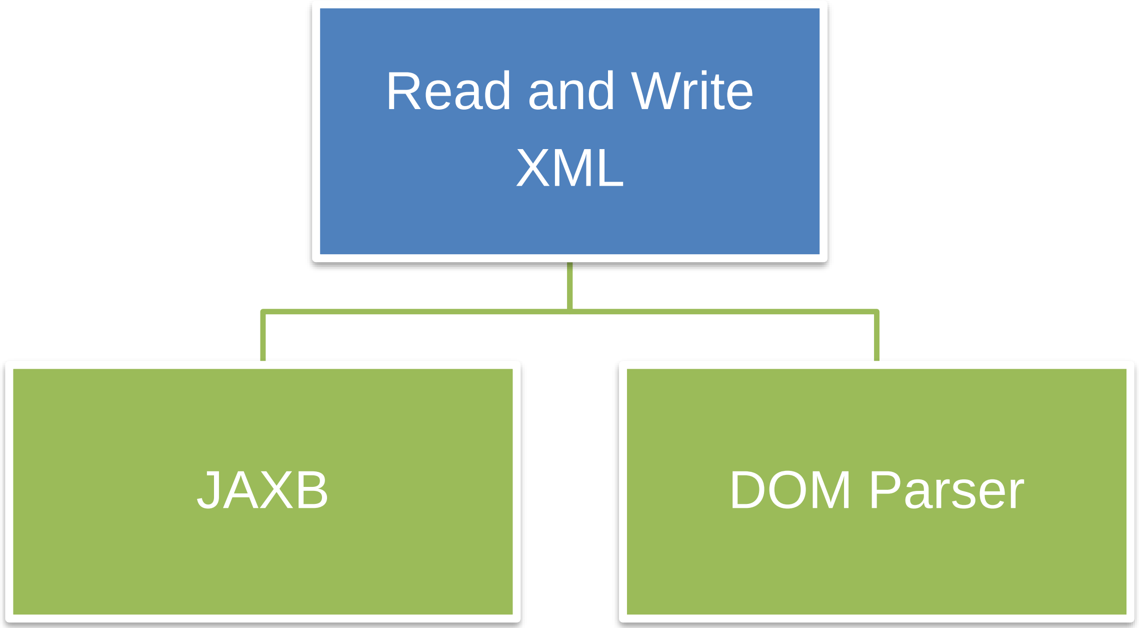
## XML Document

An XML document consists of elements (also known as tags) similar to HTML. Each element has an opening and a closing tag along with content. Every XML must have exactly one root element — one tag that wraps the remaining tags. Tag names are can-sensitive which means XML differentiates between capital and non-capital letters. Each element can have any number of nested child elements.



```xml
<?xml version="1.0" encoding="UTF-8" ?>
<user id="1">
    <name>LamNS3</name>
    <email>LamNS3@gmail.com</email>
    <roles>
        <role>Member</role>
        <role>Admin</role>
    </roles>
    <admin>true</admin>
</user>
```

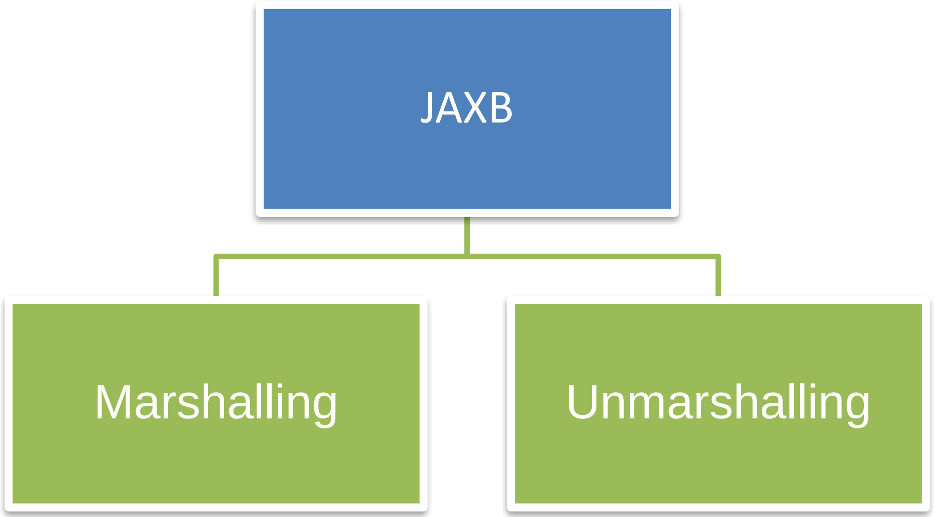
# Reading and Writing XML in Java



## Reading and Writing XML in Java

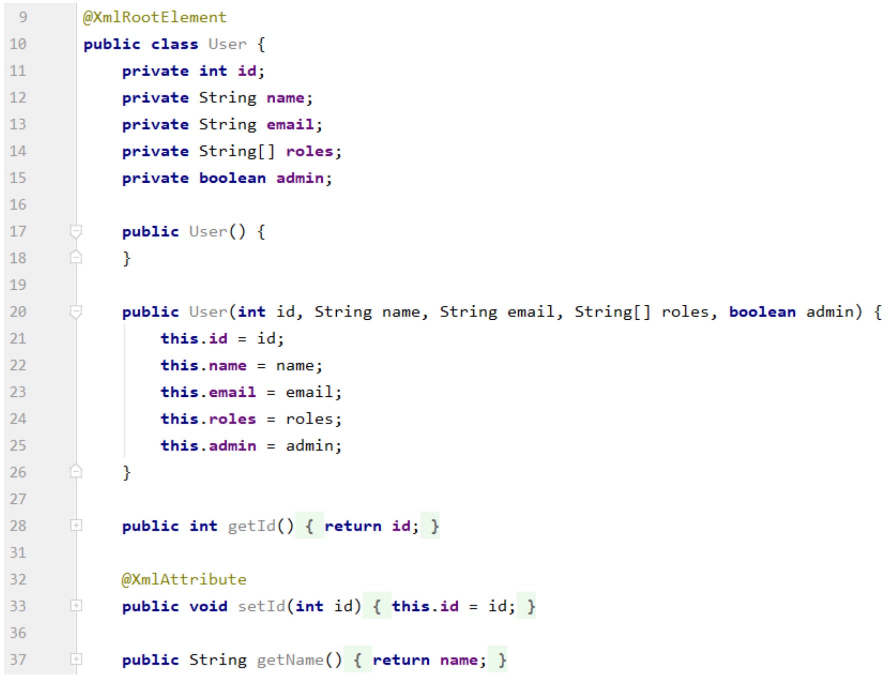
### Read and Write XML with JAXB

JAXB stands for Java Architecture for XML Binding which provides a convenient way for manipulating XML in Java. It is Java standard that defines an API for reading and writing Java objects to and from XML documents.

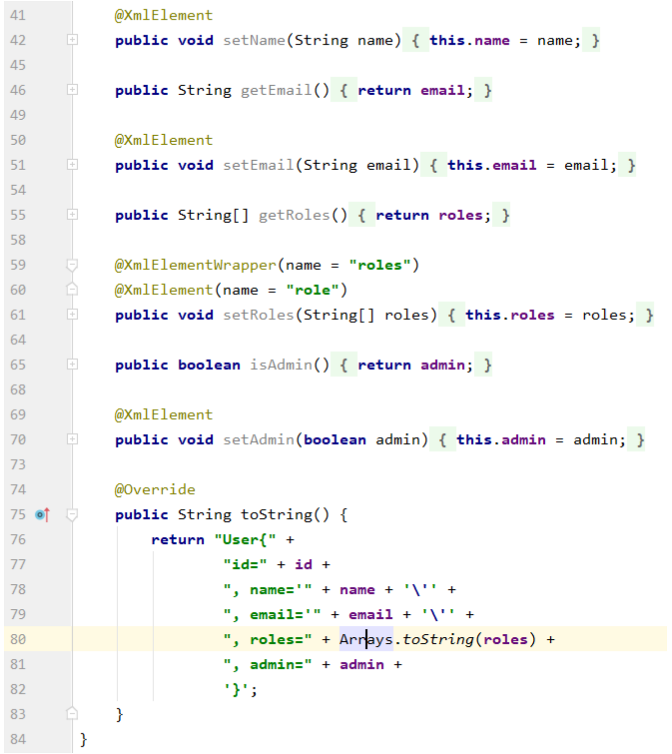


## Reading and Writing XML in Java

### Read and Write XML with JAXB



```java
@XmlRootElement
public class User {
    private int id;
    private String name;
    private String email;
    private String[] roles;
    private boolean admin;

    public User() {
    }

    public User(int id, String name, String email, String[] roles, boolean admin) {
        this.id = id;
        this.name = name;
        this.email = email;
        this.roles = roles;
        this.admin = admin;
    }

    public int getId() { return id; }

    @XmlAttribute
    public void setId(int id) { this.id = id; }

    public String getName() { return name; }
```



```java
    @XmlElement
    public void setName(String name) { this.name = name; }

    public String getEmail() { return email; }

    @XmlElement
    public void setEmail(String email) { this.email = email; }

    public String[] getRoles() { return roles; }

    @XmlElementWrapper(name = "roles")
    @XmlElement(name = "role")
    public void setRoles(String[] roles) { this.roles = roles; }

    public boolean isAdmin() { return admin; }

    @XmlElement
    public void setAdmin(boolean admin) { this.admin = admin; }

    @Override
    public String toString() {
        return "User{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", email='" + email + '\'' +
                ", roles=" + Arrays.toString(roles) +
                ", admin=" + admin +
                '}';
    }
}
```

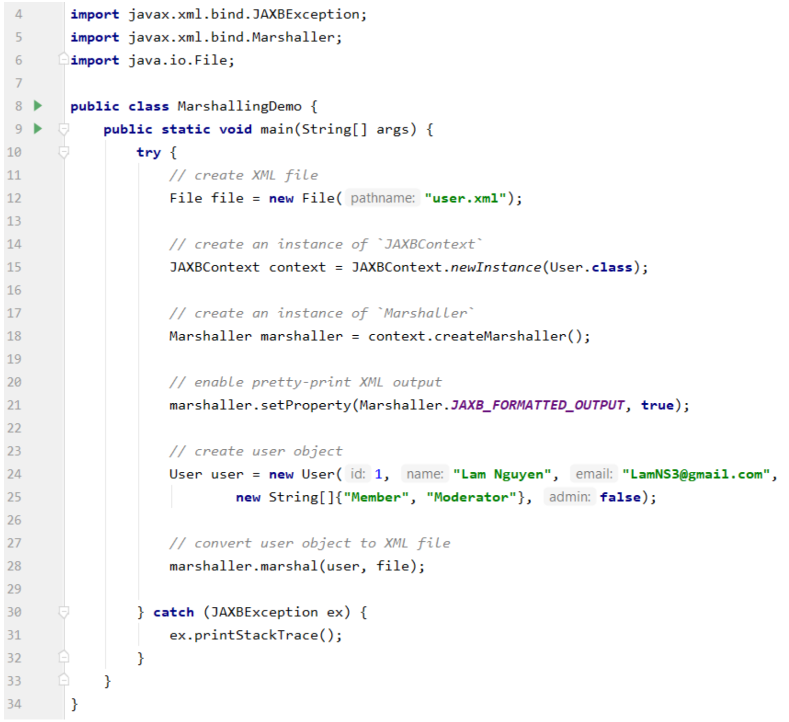
# Reading and Writing XML in Java

## Read and Write XML with JAXB

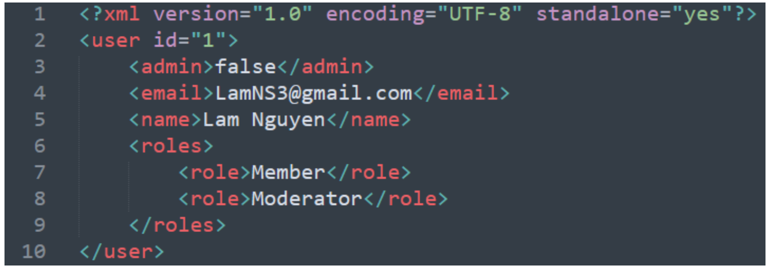
@XmlRootElement — This annotation is used to specify the root element of the XML document. It maps a class or an enum type to an XML element. By default, it uses the name of the class or enum as the name of the root element. However, you can customize the name by explicitly setting the name attribute i.e. @XmlRootElement(name = "person").
@XmlAttribute — This annotation maps a Java object property to an XML element derived from the property name. To specify a different XML property name, you can pass the name parameter to the annotation declaration.
@XmlElement — This annotation maps a Java object property to an XML element derived from the property name. The name of the XML element being mapped can be customized by using the name parameter.
@XmlElementWrapper — This annotation generates a wrapper element around the XML representation, an array of String in our case. You must explicitly specify elements of the collection by using the @XmlElement annotation.

# Reading and Writing XML in Java

## Marshalling - Convert Java Object to XML



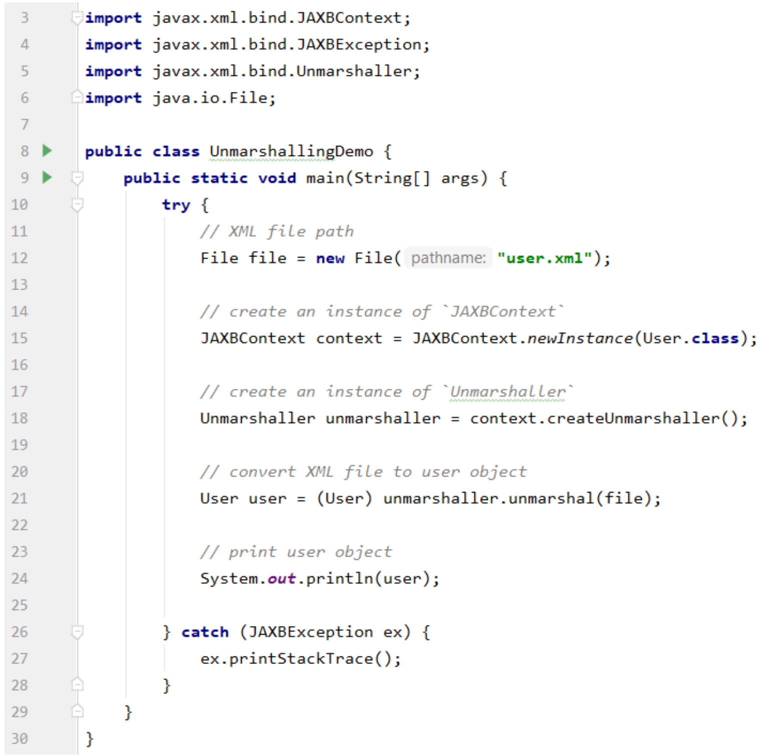
```java
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;
import java.io.File;

public class MarshallingDemo {
    public static void main(String[] args) {
        try {
            // create XML file
            File file = new File( pathname: "user.xml");

            // create an instance of `JAXBContext`
            JAXBContext context = JAXBContext.newInstance(User.class);

            // create an instance of `Marshaller`
            Marshaller marshaller = context.createMarshaller();

            // enable pretty-print XML output
            marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);

            // create user object
            User user = new User( id: 1,  name: "Lam Nguyen",  email: "LamNS3@gmail.com",
                    new String[]{"Member", "Moderator"},  admin: false);

            // convert user object to XML file
            marshaller.marshal(user, file);

        } catch (JAXBException ex) {
            ex.printStackTrace();
        }
    }
}
```



```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<user id="1">
    <admin>false</admin>
    <email>LamNS3@gmail.com</email>
    <name>Lam Nguyen</name>
    <roles>
        <role>Member</role>
        <role>Moderator</role>
    </roles>
</user>
```

## Reading and Writing XML in Java

### Unmarshalling — Convert XML to Java Object



```java
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Unmarshaller;
import java.io.File;

public class UnmarshallingDemo {
    public static void main(String[] args) {
        try {
            // XML file path
            File file = new File( pathname: "user.xml");

            // create an instance of `JAXBContext`
            JAXBContext context = JAXBContext.newInstance(User.class);

            // create an instance of `Unmarshaller`
            Unmarshaller unmarshaller = context.createUnmarshaller();

            // convert XML file to user object
            User user = (User) unmarshaller.unmarshal(file);

            // print user object
            System.out.println(user);

        } catch (JAXBException ex) {
            ex.printStackTrace();
        }
    }
}
```

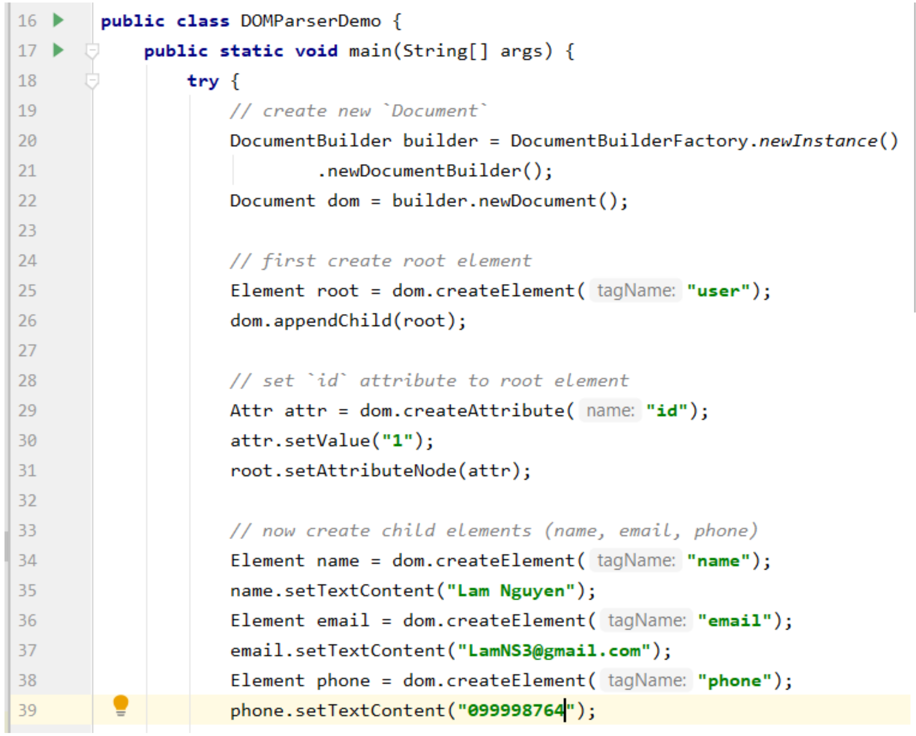
## Reading and Writing XML in Java

### Write XML to File using DOM Parser

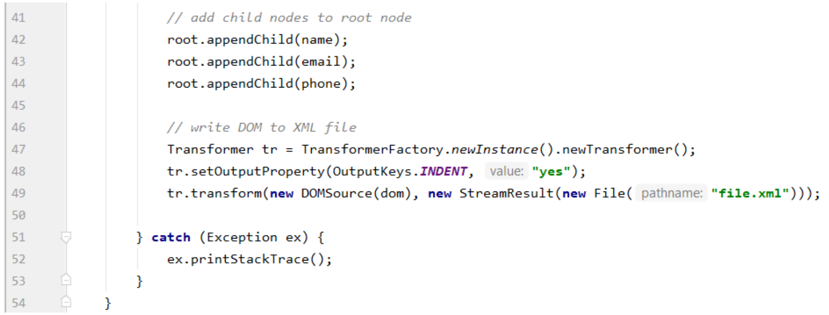
To create an XML file using the DOM parser, you have to first create an instance of Document class using DocumentBuilder. Then define all the XML content — elements, attributes, values — with Element and Attr classes. In the end, use the Transformer class to output the entire XML document to an output stream, usually a file or a string.

# Reading and Writing XML in Java

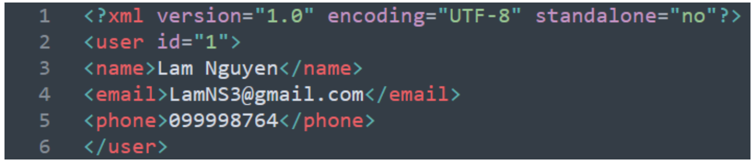
## Write XML to File using DOM Parser



```java
public class DOMParserDemo {
    public static void main(String[] args) {
        try {
            // create new `Document`
            DocumentBuilder builder = DocumentBuilderFactory.newInstance()
                    .newDocumentBuilder();
            Document dom = builder.newDocument();

            // first create root element
            Element root = dom.createElement( tagName: "user");
            dom.appendChild(root);

            // set `id` attribute to root element
            Attr attr = dom.createAttribute( name: "id");
            attr.setValue("1");
            root.setAttributeNode(attr);

            // now create child elements (name, email, phone)
            Element name = dom.createElement( tagName: "name");
            name.setTextContent("Lam Nguyen");
            Element email = dom.createElement( tagName: "email");
            email.setTextContent("LamNS3@gmail.com");
            Element phone = dom.createElement( tagName: "phone");
            phone.setTextContent("099998764");
```



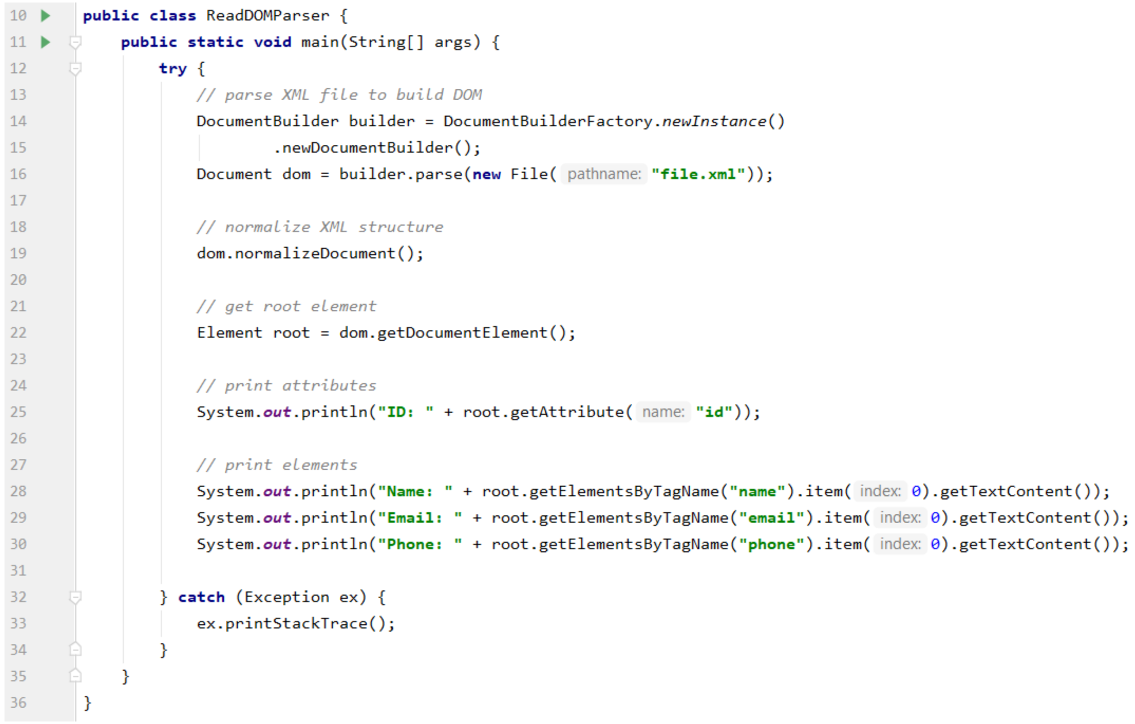
```java
            // add child nodes to root node
            root.appendChild(name);
            root.appendChild(email);
            root.appendChild(phone);

            // write DOM to XML file
            Transformer tr = TransformerFactory.newInstance().newTransformer();
            tr.setOutputProperty(OutputKeys.INDENT,  value: "yes");
            tr.transform(new DOMSource(dom), new StreamResult(new File( pathname: "file.xml")));

        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
```



```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<user id="1">
<name>Lam Nguyen</name>
<email>LamNS3@gmail.com</email>
<phone>099998764</phone>
</user>
```

# Reading and Writing XML in Java

## Read XML from File using DOM Parser



```
public class ReadDOMParser {
    public static void main(String[] args) {
        try {
            // parse XML file to build DOM
            DocumentBuilder builder = DocumentBuilderFactory.newInstance()
                    .newDocumentBuilder();
            Document dom = builder.parse(new File( pathname: "file.xml"));

            // normalize XML structure
            dom.normalizeDocument();

            // get root element
            Element root = dom.getDocumentElement();

            // print attributes
            System.out.println("ID: " + root.getAttribute( name: "id"));

            // print elements
            System.out.println("Name: " + root.getElementsByTagName("name").item( index: 0).getTextContent());
            System.out.println("Email: " + root.getElementsByTagName("email").item( index: 0).getTextContent());
            System.out.println("Phone: " + root.getElementsByTagName("phone").item( index: 0).getTextContent());

        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

# Reading and Writing XML in Java

## Summary

Although XML is not widely used as a data exchange format in modern systems, it is still used by a lot of old services on the web as a primary source of data exchange. This is also true for many file formats that store data in XML-formatted files.

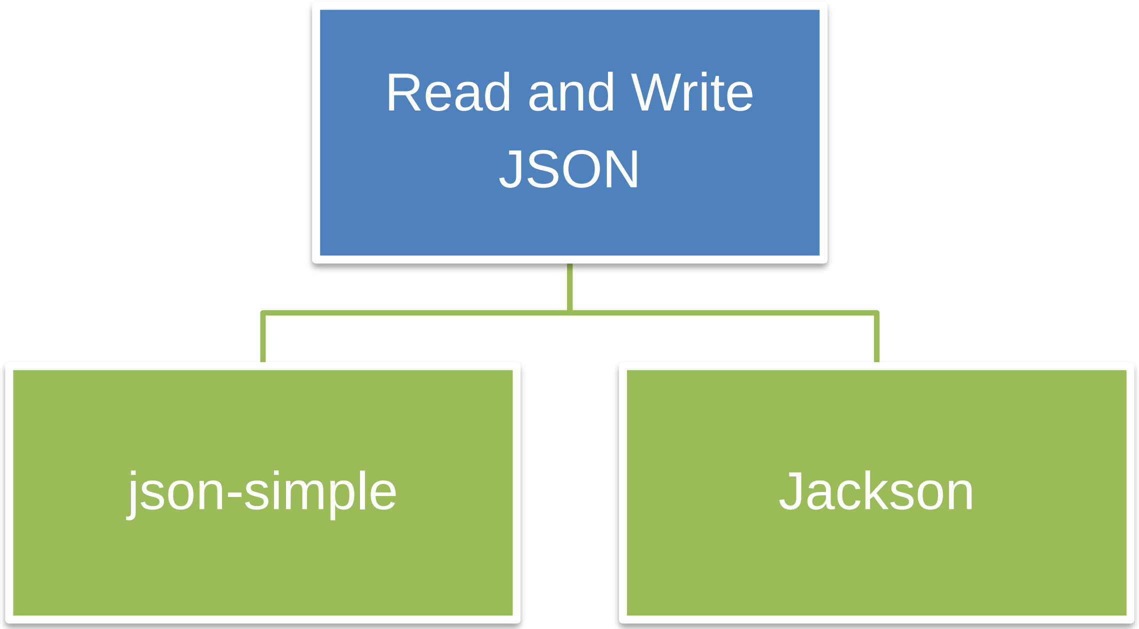
Java provides multiple ways to read and write XML files. In this article, we looked at JAXB and DOM parser for reading and writing XML data to and from a file.

JAXB is a modern replacement for old XML parsers like DOM and SAX. It provides methods to read and write Java objects to and from a file. By using JAXB annotations, we can easily define the relationship between XML elements and object attributes.

# Reading and Writing JSON in Java

JavaScript Object Notation or in short JSON is a data-interchange format that was introduced in 1999 and became widely adopted in the mid-2000s. Currently, it is the de-facto standard format for the communication between web services and their clients (browsers, mobile applications, etc.). Knowing how to read and write it is an essential skill for any software developer.

# Reading and Writing JSON in Java



## Reading and Writing JSON in Java

### Read and Write JSON with json-simple

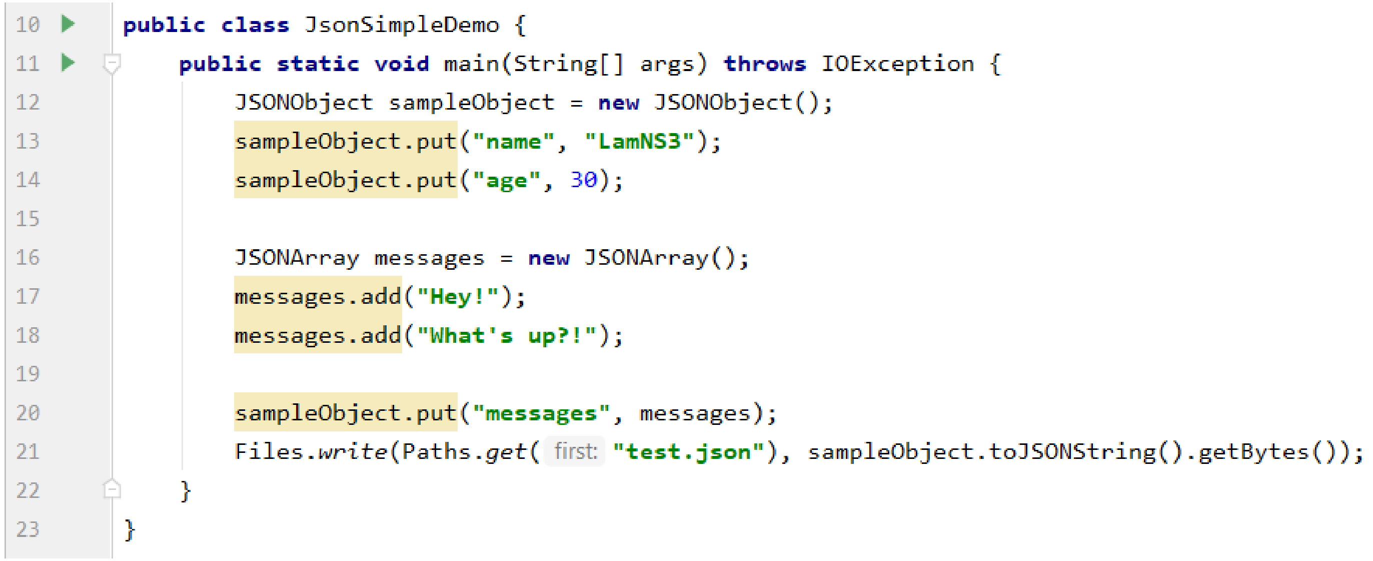
As there is no native support for JSON in Java, first of all, we should add a new dependency that would provide it for us. To begin with, we'll use the json-simple module, adding it as a Maven dependency.

```
<dependency>
   <groupId>com.googlecode.json-simple</groupId>
   <artifactId>json-simple</artifactId>
   <version>1.1.1</version>
</dependency>
```

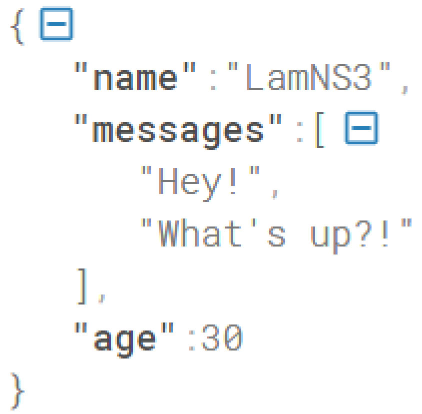
https://mvnrepository.com/artifact/com.googlecode.json-simple/json-simple/1.1.1

# Reading and Writing JSON in Java

## Read and Write JSON with json-simple



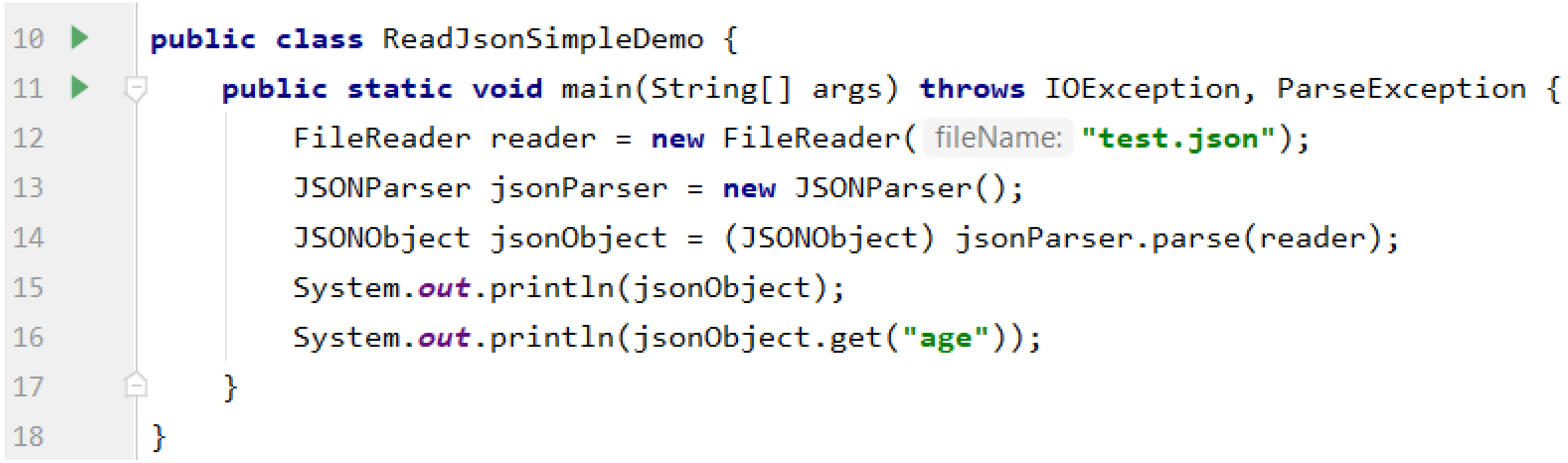
```java
public class JsonSimpleDemo {
    public static void main(String[] args) throws IOException {
        JSONObject sampleObject = new JSONObject();
        sampleObject.put("name", "LamNS3");
        sampleObject.put("age", 30);

        JSONArray messages = new JSONArray();
        messages.add("Hey!");
        messages.add("What's up?!");

        sampleObject.put("messages", messages);
        Files.write(Paths.get( first: "test.json"), sampleObject.toJSONString().getBytes());
    }
}
```



```json
{
    "name":"LamNS3",
    "messages":[
        "Hey!",
        "What's up?!"
    ],
    "age":30
}
```

## Reading and Writing JSON in Java

### Read and Write JSON with json-simple



```java
public class ReadJsonSimpleDemo {
    public static void main(String[] args) throws IOException, ParseException {
        FileReader reader = new FileReader( fileName: "test.json");
        JSONParser jsonParser = new JSONParser();
        JSONObject jsonObject = (JSONObject) jsonParser.parse(reader);
        System.out.println(jsonObject);
        System.out.println(jsonObject.get("age"));
    }
}
```

## Reading and Writing JSON in Java

### Read and Write JSON with Jackson

A library that will allow us to do all this in a very efficient manner is called Jackson. It's super common and used in big enterprise projects like Hibernate.

```
<dependency>
    <groupId>org.codehaus.jackson</groupId>
    <artifactId>jackson-mapper-asl</artifactId>
    <version>1.5.0</version>
</dependency>
```

https://mvnrepository.com/artifact/org.codehaus.jackson/jackson-mapper-asl/1.5.0

## Reading and Writing JSON in Java

### Read and Write JSON with Jackson

# Thank you