ЛАБОРАТОРНА РОБОТА № 10

Серіалізація. Логування. Документування коду. XML та JSON парсери.

Мета: практика роботи з XML та JSON парсерами, використання серіалізації, логування та докусентування коду.

Хід роботи:

Завдання 1. Створити maven Java проект java_lab_10 з пакетом com.education.ztu. Додати в проект код з лабораторної роботи №3 з пакету game. Для реалізації завдань додати необхідні залежності в файл pom.xml.

Завдання 2. Серіалізація: • Додати до сутностей в пакеті game serialVersionUID (згенерувати за допомогою Inteliij IDEA) • Виключити деякі поля з серіалізації на власний розсуд (використати ключове слово transient) • Серіалізувати та десеріалізувати сутності.

Завдання 3. Логування: • Додати логування до коду в пакеті game. Використати бібліотеки Log4J, SLF4J. • Вивести логи в консоль та в файл. • Використати різні рівні логування (trace, debug, info, warn, error, fatal)

Завдання 4. Документування коду: • Додати документаційні коментарі до коду в пакеті game • Згенерувати документацію (щоб згенерувати JavaDoc у Inteliij IDEA необхідно натиснути Tools –> Generate JavaDoc –> вказати шлях, куди зберегти документацію)

Завдання 5. XML парсери: • Реалізувати читання та збереження XML файлу використовуючи DOM парсер. • XML файл використати будь який за бажанням.

Завдання 6. JSON парсер: • Провести перетворення сутностей з Java в JSON і навпаки з JSON в Java (використайте бібліотеки Gson або Jackson) Сутності для перетворень виберіть на власний розсуд.

Лістинг програми:

XMLReader.java:

```
package com.education.ztu.Classes;

import org.w3c.dom.*;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;
import java.io.*;
import com.education.ztu.game.Student;

public class XMLWriter {
    public static void writeStudents(String filePath, Student[] students) {
        try {
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
            DocumentBuilder builder = factory.newDocumentBuilder();
            Document doc = builder.newDocument();

            Element rootElement = doc.createElement("students");
            doc.appendChild(rootElement);

            for (Student student : students) {
                Element studentElement = doc.createElement("student");
                rootElement.appendChild(studentElement);

                Element name = doc.createElement("name");
                name.appendChild(doc.createTextNode(student.getName()));
                studentElement.appendChild(name);

                Element age = doc.createElement("age");

age.appendChild(doc.createTextNode(String.valueOf(student.getAge())));
                studentElement.appendChild(age);
            }

            TransformerFactory transformerFactory =
TransformerFactory.newInstance();
            Transformer transformer = transformerFactory.newTransformer();
            transformer.setOutputProperty(OutputKeys.INDENT, "yes");

            DOMSource source = new DOMSource(doc);

            File dir = new File("data");
            if (!dir.exists()) {
                dir.mkdir();
            }

            StreamResult result = new StreamResult(new File(filePath));
            transformer.transform(source, result);

            System.out.println("XML файл успішно збережено в папку 'data'!");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

XMLWriter.java:

```java
package com.education.ztu.Classes;

import org.w3c.dom.*;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;
import java.io.*;
import com.education.ztu.game.Student;

public class XMLWriter {
    public static void writeStudents(String filePath, Student[] students) {
        try {
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
            DocumentBuilder builder = factory.newDocumentBuilder();
            Document doc = builder.newDocument();

            Element rootElement = doc.createElement("students");
            doc.appendChild(rootElement);

            for (Student student : students) {
                Element studentElement = doc.createElement("student");
                rootElement.appendChild(studentElement);

                Element name = doc.createElement("name");
                name.appendChild(doc.createTextNode(student.getName()));
                studentElement.appendChild(name);

                Element age = doc.createElement("age");

age.appendChild(doc.createTextNode(String.valueOf(student.getAge())));
                studentElement.appendChild(age);
            }

            TransformerFactory transformerFactory =
TransformerFactory.newInstance();
            Transformer transformer = transformerFactory.newTransformer();
            transformer.setOutputProperty(OutputKeys.INDENT, "yes");

            DOMSource source = new DOMSource(doc);

            File dir = new File("data");
            if (!dir.exists()) {
                dir.mkdir();
            }

            StreamResult result = new StreamResult(new File(filePath));
            transformer.transform(source, result);

            System.out.println("XML файл успішно збережено в папку 'data'!");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

AgeComparator.java:

```java
/**
 * A comparator implementation for comparing two Participant objects by their age.
 * This class can be used to sort Participant objects in ascending order of age.
 */
package com.education.ztu.game;
```

```java
import java.util.Comparator;

public class AgeComparator implements Comparator<Participant> {

    /**
     * Compares two Participant objects based on their age.
     *
     * @param p1 the first Participant object to compare
     * @param p2 the second Participant object to compare
     * @return a negative integer, zero, or a positive integer as the first
argument
     *         is less than, equal to, or greater than the second.
     */
    @Override
    public int compare(Participant p1, Participant p2) {
        return Integer.compare(p1.getAge(), p2.getAge());
    }
}
```

Employee.java:

```java
/**
 * Represents an Employee who is also a Participant in the game.
 * This class extends the Participant class, inheriting its properties and
behavior.
 */
package com.education.ztu.game;

public class Employee extends Participant {

    /**
     * Constructs an Employee object with a specified name and age.
     *
     * @param name the name of the employee
     * @param age  the age of the employee
     */
    public Employee(String name, int age) {
        super(name, age);
    }
}
```

Game.java:

```java
/**
 * Demonstrates the creation and management of teams in a game,
 * with participants categorized as Scholars, Students, or Employees.
 * This program showcases how teams can be created, participants added, and
competitions held.
 */
package com.education.ztu.game;

public class Game {

    /**
     * The entry point of the application.
     *
     * @param args command-line arguments (not used in this application)
     */
    public static void main(String[] args) {

        // Creating Scholar participants
        Schoolar schoolar1 = new Schoolar("Ivan", 13);
        Schoolar schoolar2 = new Schoolar("Mariya", 15);
```

```
        // Creating Student participants
        Student student1 = new Student("Mykola", 20);
        Student student2 = new Student("Viktoria", 21);

        // Creating Employee participants
        Employee employee1 = new Employee("Andriy", 28);
        Employee employee2 = new Employee("Oksana", 25);

        // Creating and populating a team of Scholars
        Team<Schoolar> scollarTeam = new Team<>("Dragon");
        scollarTeam.addNewParticipant(schoolar1);
        scollarTeam.addNewParticipant(schoolar2);

        // Creating and populating a team of Students
        Team<Student> studentTeam = new Team<>("Vpered");
        studentTeam.addNewParticipant(student1);
        studentTeam.addNewParticipant(student2);

        // Creating and populating a team of Employees
        Team<Employee> employeeTeam = new Team<>("Robotyagi");
        employeeTeam.addNewParticipant(employee1);
        employeeTeam.addNewParticipant(employee2);

        // Creating a second team of Scholars and adding participants
        Team<Schoolar> scollarTeam2 = new Team<>("Rozumnyky");
        Schoolar schoolar3 = new Schoolar("Sergey", 12);
        Schoolar schoolar4 = new Schoolar("Olga", 14);
        scollarTeam2.addNewParticipant(schoolar3);
        scollarTeam2.addNewParticipant(schoolar4);

        // Initiating a competition between two teams of Scholars
        scollarTeam.playWith(scollarTeam2);
    }
}
```

GameEntity.java:

```
/**
 * Represents a game entity with attributes such as name, level, health, and
weapon.
 * Implements the Serializable interface to allow the object to be serialized,
 * while excluding the health attribute from serialization using the transient
keyword.
 */
package com.education.ztu.game;

import java.io.Serializable;

public class GameEntity implements Serializable {
    private static final long serialVersionUID = 1L; // Ensures compatibility
during serialization

    private String name;       // Name of the game entity
    private int level;         // Level of the game entity
    private transient int health; // Health of the game entity (not serialized)
    private String weapon;     // Weapon used by the game entity

    /**
     * Constructs a GameEntity object with the specified attributes.
     *
     * @param name   the name of the entity
     * @param level  the level of the entity
```

```
     * @param health the health of the entity (not serialized)
     * @param weapon the weapon of the entity
     */
    public GameEntity(String name, int level, int health, String weapon) {
        this.name = name;
        this.level = level;
        this.health = health;
        this.weapon = weapon;
    }

    /**
     * Returns a string representation of the GameEntity object.
     *
     * @return a string containing the entity's attributes
     */
    @Override
    public String toString() {
        return "GameEntity{" +
                "name='" + name + '\'' +
                ", level=" + level +
                ", health=" + health +
                ", weapon='" + weapon + '\'' +
                '}';
    }
}
```

NameAndAgeComparator.java:

```
/**
 * Provides a comparator for comparing Participant objects based on their name and
age.
 * The comparison is done first by name in lexicographical order, then by age in
ascending order.
 */
package com.education.ztu.game;

import java.util.Comparator;

public class NameAndAgeComparator {

    /**
     * Returns a Comparator that compares Participant objects by their name and
age.
     *
     * @return a Comparator object for comparing Participant instances
     */
    public static Comparator<Participant> getComparator() {
        return Comparator.comparing(Participant::getName)  // First, compare by
name
                .thenComparing(Participant::getAge);     // Then, compare by age
if names are the same
    }
}
```

Participant.java:

```
/**
 * Represents a participant in the game with a name and age.
 * This class implements Cloneable to allow object cloning and Comparable for
natural sorting by name.
 * Subclasses should provide specific behavior for participants in the game.
```

```
 */
package com.education.ztu.game;

public abstract class Participant implements Cloneable, Comparable<Participant> {
    private String name;  // Name of the participant
    private int age;      // Age of the participant
    public Participant() {

    }
    /**
     * Constructs a Participant object with a specified name and age.
     *
     * @param name the name of the participant
     * @param age  the age of the participant
     */
    public Participant(String name, int age) {
        this.name = name;
        this.age = age;
    }

    /**
     * Returns the name of the participant.
     *
     * @return the name of the participant
     */
    public String getName() {
        return name;
    }

    /**
     * Returns the age of the participant.
     *
     * @return the age of the participant
     */
    public int getAge() {
        return age;
    }

    /**
     * Sets the name of the participant.
     *
     * @param name the name to set
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * Sets the age of the participant.
     *
     * @param age the age to set
     */
    public void setAge(int age) {
        this.age = age;
    }

    /**
     * Compares this participant to another participant based on their name.
     * The comparison is lexicographical based on the participant's name.
     *
     * @param other the other participant to compare to
     * @return a negative integer, zero, or a positive integer if this
participant's name is less than, equal to, or greater than the other's name
```

```
     */
    @Override
    public int compareTo(Participant other) {
        return this.name.compareTo(other.name);
    }

    /**
     * Creates and returns a clone of this participant.
     *
     * @return a clone of this participant
     * @throws AssertionError if cloning is not supported
     */
    @Override
    public Participant clone() {
        try {
            return (Participant) super.clone();
        } catch (CloneNotSupportedException e) {
            throw new AssertionError("Cloning not supported");
        }
    }

    /**
     * Determines if this participant is equal to another object.
     * Two participants are considered equal if their names and ages are the same.
     *
     * @param obj the object to compare to
     * @return true if the participants are equal, false otherwise
     */
    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return false;

        Participant that = (Participant) obj;
        return age == that.age && name.equals(that.name);
    }

    /**
     * Returns the hash code of this participant.
     * The hash code is generated based on the participant's name and age.
     *
     * @return the hash code of this participant
     */
    @Override
    public int hashCode() {
        int result = name.hashCode();
        result = 31 * result + age;
        return result;
    }

    /**
     * Returns a string representation of the participant, including their name
and age.
     *
     * @return a string representation of the participant
     */
    @Override
    public String toString() {
        return "Participant{name='" + name + "', age=" + age + "}";
    }
}
```

Schoolar.java:

```java
/**
 * Represents a Schoolar participant in the game, extending the Participant class.
 * Inherits the name and age properties from the Participant class.
 */
package com.education.ztu.game;

public class Schoolar extends Participant {

    public Schoolar() {
    }
    /**
     * Constructs a Schoolar object with a specified name and age.
     * This constructor calls the constructor of the Participant class.
     *
     * @param name the name of the Schoolar
     * @param age  the age of the Schoolar
     */
    public Schoolar(String name, int age) {
        super(name, age);  // Викликає конструктор батьківського класу Participant
    }
}
```

Student.java:

```java
/**
 * Represents a Student participant in the game, extending the Participant class.
 * Inherits the name and age properties from the Participant class.
 */
package com.education.ztu.game;

public class Student extends Participant {

    /**
     * Constructs a Student object with a specified name and age.
     * This constructor calls the constructor of the Participant class.
     *
     * @param name the name of the Student
     * @param age  the age of the Student
     */
    public Student(String name, int age) {
        super(name, age);
    }
}
```

Team.java:

```java
/**
 * Represents a Student participant in the game, extending the Participant class.
 * Inherits the name and age properties from the Participant class.
 */
package com.education.ztu.game;

public class Student extends Participant {

    /**
     * Constructs a Student object with a specified name and age.
     * This constructor calls the constructor of the Participant class.
     *
     * @param name the name of the Student
```

```
     * @param age  the age of the Student
     */
    public Student(String name, int age) {
        super(name, age);
    }
}
```

Main.java:

```
package com.education.ztu;

import com.education.ztu.game.GameEntity;
import java.io.*;

public class Main {
    public static void main(String[] args) {
        GameEntity entity = new GameEntity("Knight", 5, 100, "Sword");

        String filePath = "entity.ser";

        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(filePath))) {
            oos.writeObject(entity);
            System.out.println("Entity serialized successfully.");
        } catch (IOException e) {
            System.out.println("Serialization error: " + e.getMessage());
        }

        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(filePath))) {
            GameEntity deserializedEntity = (GameEntity) ois.readObject();
            System.out.println("Deserialized entity: " + deserializedEntity);
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Deserialization error: " + e.getMessage());
        }
    }
}
```

Main2.java:

```
package com.education.ztu;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class Main2 {
    private static final Logger logger = LogManager.getLogger(Main2.class);

    public static void main(String[] args) {
        logger.info("Application started");
        logger.fatal("Fatal message - application might crash");
        logger.error("Error message - something went wrong");
        logger.warn("Warning message - be careful");
        logger.info("Info message - just FYI");
        logger.debug("Debug message - useful for debugging");
        logger.trace("Trace message - very detailed info");

        try {
            int result = divide(10, 2);
            logger.debug("Division result: {}", result);
        } catch (Exception e) {
```

```
            logger.error("Error occurred during division", e);
        }

        logger.info("Application finished");
    }

    private static int divide(int a, int b) {
        logger.trace("Entering divide() with a = {} and b = {}", a, b);
        if (b == 0) {
            logger.warn("Division by zero attempted!");
            throw new ArithmeticException("Cannot divide by zero");
        }
        int result = a / b;
        logger.trace("Exiting divide() with result = {}", result);
        return result;
    }
}
```

```
package com.education.ztu;

import com.education.ztu.Classes.XMLReader;
import com.education.ztu.Classes.XMLWriter;
import com.education.ztu.game.Student;

public class Main3 {

    public static void main(String[] args) {

        Student[] students = {
                new Student("Mark Spencer", 23),
                new Student("Emily Johnson", 24)
        };

        XMLWriter.writeStudents("data/students.xml", students);

        String filePath = "data/students.xml";

        Student[] readStudents = XMLReader.readStudents(filePath);

        if (readStudents != null) {
            for (Student student : readStudents) {
                System.out.println("Name: " + student.getName() + ", Age: " +
student.getAge());
            }
        } else {
            System.out.println("Не вдалося зчитати студентів.");
        }
    }
}
```

Main3.java:

```
package com.education.ztu;

import com.education.ztu.Classes.XMLReader;
import com.education.ztu.Classes.XMLWriter;
import com.education.ztu.game.Student;

public class Main3 {

    public static void main(String[] args) {
```

```
        Student[] students = {
                new Student("Mark Spencer", 23),
                new Student("Emily Johnson", 24)
        };

        XMLWriter.writeStudents("data/students.xml", students);

        String filePath = "data/students.xml";

        Student[] readStudents = XMLReader.readStudents(filePath);

        if (readStudents != null) {
            for (Student student : readStudents) {
                System.out.println("Name: " + student.getName() + ", Age: " +
student.getAge());
            }
        } else {
            System.out.println("Не вдалося зчитати студентів.");
        }
    }
}
```

Main4.java:

```
package com.education.ztu;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.education.ztu.game.Schoolar;

public class Main4 {

    public static void main(String[] args) throws Exception {

        Schoolar person = new Schoolar("John Doe", 30);

        ObjectMapper objectMapper = new ObjectMapper();
        String json = objectMapper.writeValueAsString(person);
        System.out.println("Java to JSON: " + json);

        String jsonInput = "{\"name\":\"Jane Doe\",\"age\":30}";
        Schoolar deserializedPerson = objectMapper.readValue(jsonInput,
Schoolar.class);
        System.out.println("JSON to Java: Name - " + deserializedPerson.getName()
+ ", Age - " + deserializedPerson.getAge());
    }
}
```

log4j2.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
    <Appenders>
        <!-- Консольний аппендер -->
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %c{1}.%M [%L] %-5level -
%msg%n"/>
        </Console>

        <!-- Файл для Info -->
        <File name="InfoFile" fileName="data/info.log">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %c{1}.%M [%L] %-5level -
%msg%n"/>
```

```
            <Filters>
                <ThresholdFilter level="info" onMatch="ACCEPT" onMismatch="DENY"/>
            </Filters>
        </File>

        <File name="DebugFile" fileName="data/debug.log">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %c{1}.%M [%L] %-5level -
%msg%n"/>
            <Filters>
                <ThresholdFilter level="debug" onMatch="ACCEPT"
onMismatch="DENY"/>
            </Filters>
        </File>

        <File name="ErrorFile" fileName="data/error.log">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %c{1}.%M [%L] %-5level -
%msg%n"/>
            <Filters>
                <ThresholdFilter level="error" onMatch="ACCEPT"
onMismatch="DENY"/>
            </Filters>
        </File>
    </Appenders>

    <Loggers>
        <Logger name="com.education.ztu.Main2" level="trace" additivity="false">
            <AppenderRef ref="Console"/>
            <AppenderRef ref="InfoFile"/>
            <AppenderRef ref="DebugFile"/>
            <AppenderRef ref="ErrorFile"/>
        </Logger>

        <Root level="warn">
            <AppenderRef ref="Console"/>
            <AppenderRef ref="ErrorFile"/>
        </Root>
    </Loggers>
</Configuration>
```

Pom.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>Lab_10</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>23</maven.compiler.source>
        <maven.compiler.target>23</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-core</artifactId>
```

```
            <version>2.21.0</version>
        </dependency>

        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-api</artifactId>
            <version>2.21.0</version>
        </dependency>

        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>2.18.2</version>
        </dependency>


    </dependencies>

</project>
```

Результат програми:

```
"C:\Program Files\Java\jdk-25\bin\java.exe" "-javaagent:C:\Program Files\JetBrains
Entity serialized successfully.
Deserialized entity: GameEntity{name='Knight', level=5, health=0, weapon='Sword'}

Process finished with exit code 0
```

Рис.1 Результат Main.java

Main2

```
"C:\Program Files\Java\jdk-25\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\Int
21:45:29.456 [main] Main2.main [10] INFO  - Application started
21:45:29.459 [main] Main2.main [11] FATAL - Fatal message - application might crash
21:45:29.460 [main] Main2.main [12] ERROR - Error message - something went wrong
21:45:29.460 [main] Main2.main [13] WARN  - Warning message - be careful
21:45:29.461 [main] Main2.main [14] INFO  - Info message - just FYI
21:45:29.461 [main] Main2.main [15] DEBUG - Debug message - useful for debugging
21:45:29.462 [main] Main2.main [16] TRACE - Trace message - very detailed info
21:45:29.470 [main] Main2.divide [29] TRACE - Entering divide() with a = 10 and b = 2
21:45:29.470 [main] Main2.divide [35] TRACE - Exiting divide() with result = 5
21:45:29.470 [main] Main2.main [20] DEBUG - Division result: 5
21:45:29.470 [main] Main2.main [25] INFO  - Application finished

Process finished with exit code 0
```

Рис.2 Результат Main2.java

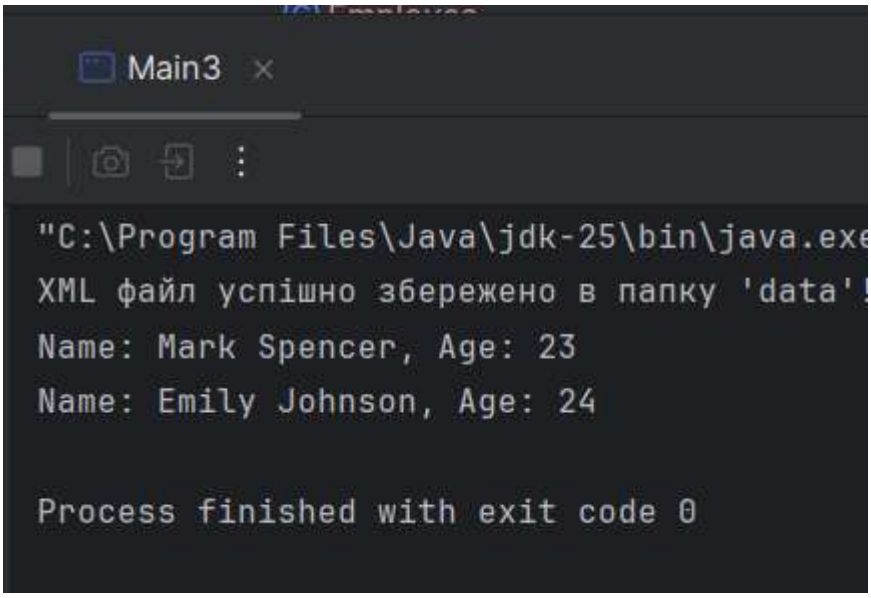

Рис.3 Результат Main3.java

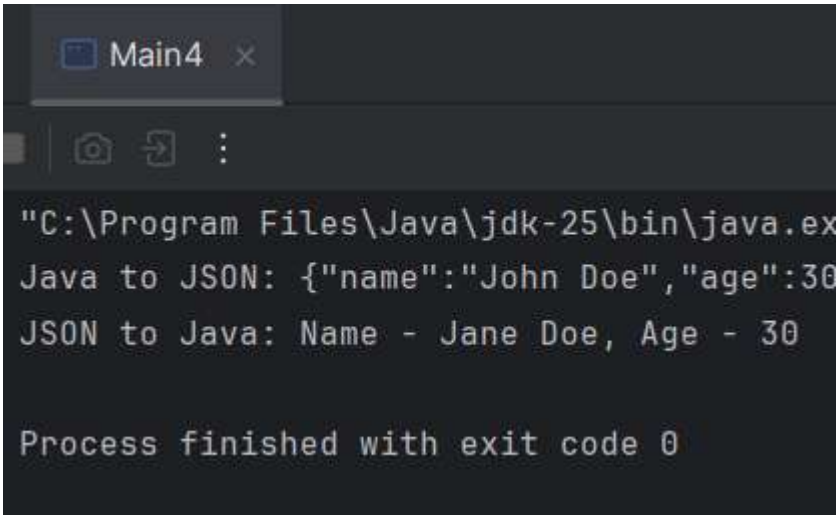

Рис.4 Результат Main4.java

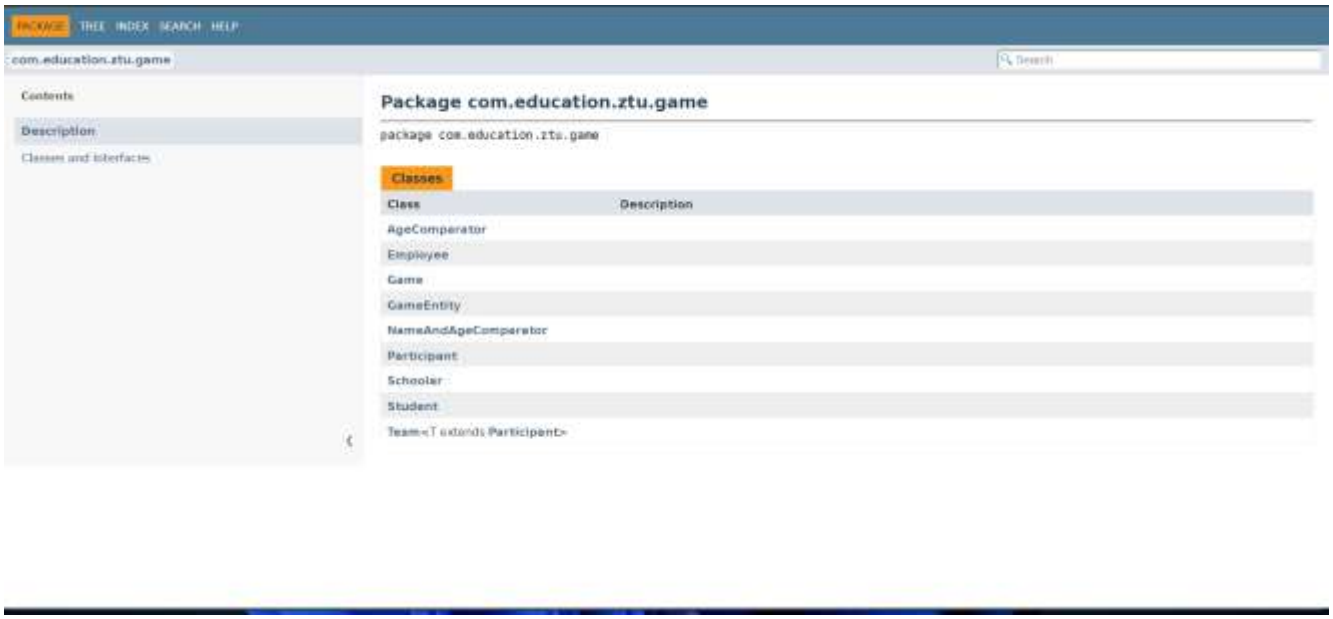

Рис.5 Документація з класами

Рис.6 Документація класу Hierarchy

Рис.7 Документація Index

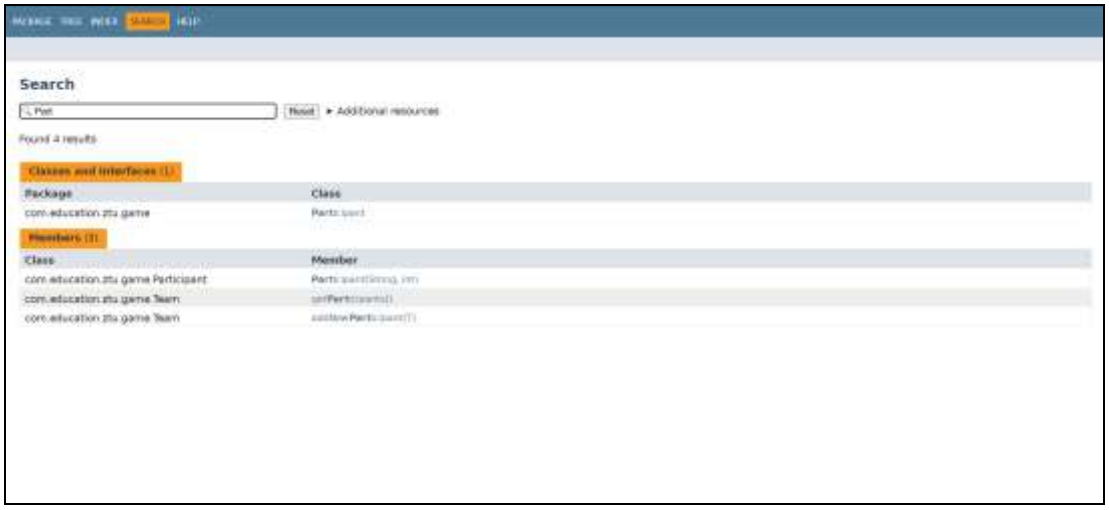

Рис.8 Документація з Search

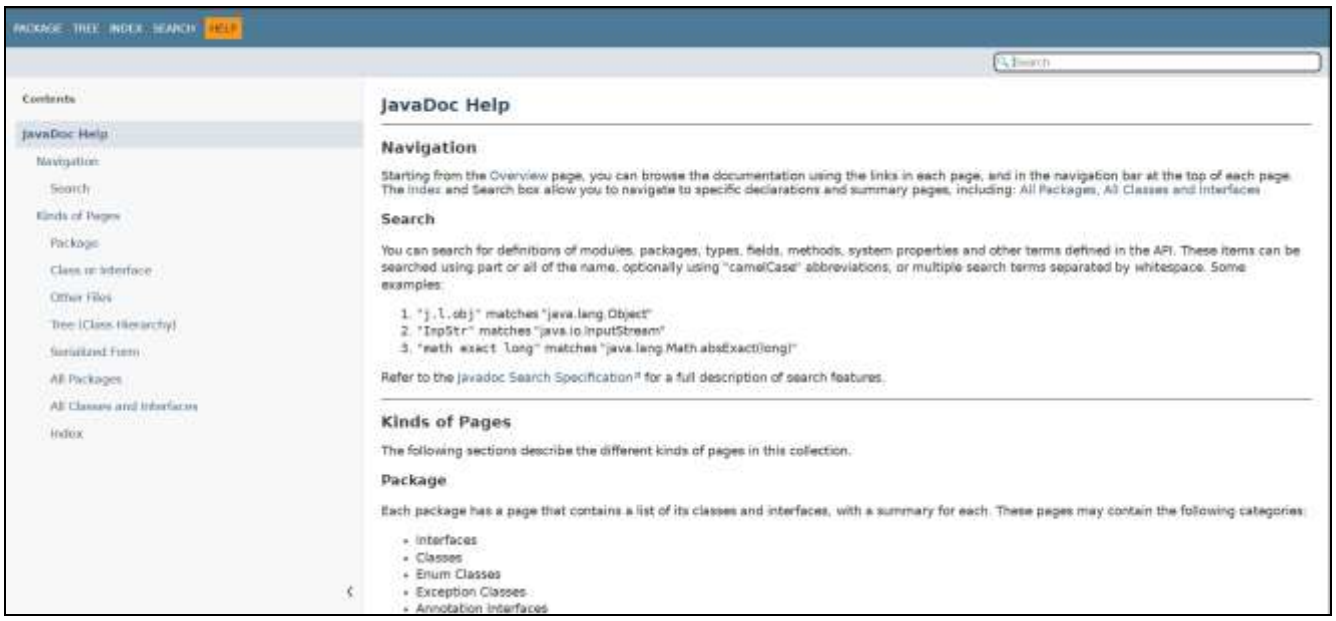

Рис.9 Документація з Help

Посилання на репозиторій: https://github.com/Sasha1845/Java

Висновок: Я навчився роботи з XML та JSON парсерами, використання серіалізації, логування та докусентування коду.