Лабораторна робота №10

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

Мета роботи: 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:

Директорія Game:

Emloyee.java

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

/**
 * Represents an employee/working professional participant in the game system.
 *
 * <p>This class extends the abstract {@link Participant} class and specializes it
for
 * working professionals and employees, typically ages 25 and above. Employee
participants
 * form teams of their own type and compete with other employee teams.</p>
 *
 * <h2>Characteristics:</h2>
 * <ul>
 *   <li><b>Age Range:</b> Typically 25+ years old (working professionals)</li>
 *   <li><b>Team Composition:</b> Teams can only contain other Employee
participants</li>
 *   <li><b>Serialization:</b> Fully serializable with custom
serialVersionUID</li>
 * </ul>
 *
 * <h2>Type Safety:</h2>
 * <p>Through Java Generics in the {@link Team} class, a {@code Team<Employee>}
can only
 * contain Employee participants. This prevents mixing different participant types
in
 * the same team and is enforced at compile time. For example, you cannot add a
Student
 * or Schoolar to an Employee team.</p>
 *
 * <h2>Usage Example:</h2>
 * <pre>
 * // Create individual employee participants
 * Employee andriy = new Employee("Andriy", 28);
 * Employee oksana = new Employee("Oksana", 25);
 *
 * // Create a team for employees
 * Team&lt;Employee&gt; robotyagiTeam = new Team&lt;&gt;("Robotyagi");
 *
 * // Add employees to the team
 * robotyagiTeam.addNewParticipant(andriy);
 * robotyagiTeam.addNewParticipant(oksana);
 *
 * // Create another team and play a game
 * Team&lt;Employee&gt; professionalsTeam = new Team&lt;&gt;("Professionals");
 * robotyagiTeam.playWith(professionalsTeam);
 * </pre>
 *
 * <h2>Inherited Features:</h2>
 * <p>Employee inherits all functionality from Participant including:
 * <ul>
```

```
 *   <li>Name and age management ({@link #getName()}, {@link #getAge()})</li>
 *   <li>Cloning support ({@link #clone()})</li>
 *   <li>Serialization support</li>
 *   <li>Natural ordering by name ({@link #compareTo(Participant)})</li>
 *   <li>Equality and hashing ({@link #equals(Object)}, {@link #hashCode()})</li>
 * </ul>
 * </p>
 *
 * <h2>Differences from Other Participant Types:</h2>
 * <p>
 * While Schoolar represents school-age students and Student represents university
 * students, Employee represents working professionals with their own experience
 * level and professional characteristics.
 * </p>
 *
 * @author ZTU Student
 * @version 1.0
 * @since 2024
 *
 * @see Participant
 * @see Schoolar
 * @see Student
 * @see Team
 * @see Game
 */
public class Employee extends Participant {

    /**
     * Serial version UID for serialization.
     * <p>This unique identifier ensures that Employee objects can be safely
     * serialized and deserialized across different versions of the class.</p>
     */
    private static final long serialVersionUID = 3333333333333333333L;

    /**
     * Constructs a new Employee participant with the specified name and age.
     *
     * <p>This constructor delegates to the parent {@link Participant} class
     * constructor, which initializes the name, age, and temporary
information.</p>
     *
     * @param name the name of the employee (e.g., "Andriy", "Oksana")
     * @param age the age of the employee in years (typically 25+)
     *
     * @throws NullPointerException if name is null
     *
     * @see Participant#Participant(String, int)
     */
    public Employee(String name, int age) {
        super(name, age);
    }

    /**
     * Returns a string representation of this Employee participant.
     *
     * <p>The string includes the employee's name, age, and temporary information.
     * Format: {@code Employee{name='...', age=..., tempInfo='...'}}</p>
     *
     * <p>This method overrides the parent {@link Participant#toString()} to
provide
     * an Employee-specific string representation.</p>
     *
     * @return a string representation containing {@code Employee{name, age,
```

```
tempInfo}}
     */
    @Override
    public String toString() {
        return "Employee{" +
                "name='" + getName() + '\'' +
                ", age=" + getAge() +
                ", tempInfo='" + getTempInfo() + '\'' +
                '}';
    }
}
```

Game.java

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

/**
 * Demonstration class for the game system with type-safe team management.
 *
 * <p>This class serves as the main entry point and example usage of the game
framework.
 * It demonstrates how to:
 * <ul>
 *   <li>Create participants of different types (Schoolar, Student, Employee)</li>
 *   <li>Form teams with type-safe generics</li>
 *   <li>Simulate games between teams</li>
 * </ul>
 * </p>
 *
 * <h2>Program Structure:</h2>
 * <p>The program creates three leagues:
 * <ul>
 *   <li><b>Schoolar League:</b> Two teams with schoolar participants
 *     <ul>
 *       <li>Dragon team (Ivan, Mariya)</li>
 *       <li>Rozumnyky team (Sergey, Olga)</li>
 *     </ul>
 *   </li>
 *   <li><b>Student League:</b> Two teams with student participants
 *     <ul>
 *       <li>Vpered team (Mykola, Viktoria)</li>
 *       <li>Intellect team (Andriy, Kateryna)</li>
 *     </ul>
 *   </li>
 *   <li><b>Employee League:</b> Two teams with employee participants
 *     <ul>
 *       <li>Robotyagi team (Andriy, Oksana)</li>
 *       <li>Professionals team (Petro, Natalia)</li>
 *     </ul>
 *   </li>
 * </ul>
 * </p>
 *
 * <h2>Type Safety Guarantees:</h2>
 * <p>Thanks to Java Generics, the following constraints are enforced at compile
time:
 * <ul>
 *   <li>Cannot add a Student to a {@code Team<Schoolar>}</li>
 *   <li>Cannot add an Employee to a {@code Team<Student>}</li>
 *   <li>Cannot mix different participant types in the same team</li>
 *   <li>Teams of different types cannot play against each other</li>
 * </ul>
 * These constraints prevent logical errors that would only be caught at runtime
 * in a non-generic implementation.
```

```
 * </p>
 *
 * <h2>Game Simulation:</h2>
 * <p>Each team plays one game:
 * <ul>
 *   <li>Schoolar teams: Dragon vs Rozumnyky</li>
 *   <li>Student teams: Vpered vs Intellect</li>
 *   <li>Employee teams: Robotyagi vs Professionals</li>
 * </ul>
 * The winner of each game is determined randomly.
 * </p>
 *
 * <h2>Execution:</h2>
 * <p>Run this class as a Java application to see the game demonstration:
 * <pre>
 * java com.education.ztu.game.Game
 * </pre>
 * </p>
 *
 * <h2>Output:</h2>
 * <p>The program outputs:
 * <ul>
 *   <li>Creation messages for each team and participant</li>
 *   <li>Game results for each match</li>
 *   <li>Notifications about type safety guarantees</li>
 * </ul>
 * </p>
 *
 * @author ZTU Student
 * @version 1.0
 * @since 2024
 *
 * @see Participant
 * @see Schoolar
 * @see Student
 * @see Employee
 * @see Team
 */
public class Game {

    /**
     * Main method to run the game demonstration.
     *
     * <p>This method:
     * <ul>
     *   <li>Creates 12 participants (4 of each type)</li>
     *   <li>Organizes them into 6 teams (2 of each type)</li>
     *   <li>Simulates 3 games (one for each participant type)</li>
     *   <li>Displays team creation messages and game results</li>
     * </ul>
     * </p>
     *
     * @param args command-line arguments (not used)
     */
    public static void main(String[] args) {
        Schoolar schoolar1 = new Schoolar("Ivan", 13);
        Schoolar schoolar2 = new Schoolar("Mariya", 15);
        Schoolar schoolar3 = new Schoolar("Sergey", 12);
        Schoolar schoolar4 = new Schoolar("Olga", 14);

        Student student1 = new Student("Mykola", 20);
        Student student2 = new Student("Viktoria", 21);
        Student student3 = new Student("Andriy", 22);
```

```
        Student student4 = new Student("Kateryna", 19);

        Employee employee1 = new Employee("Andriy", 28);
        Employee employee2 = new Employee("Oksana", 25);
        Employee employee3 = new Employee("Petro", 30);
        Employee employee4 = new Employee("Natalia", 27);

        System.out.println("=== Creating Schoolar Teams ===");
        Team<Schoolar> schoolarTeam1 = new Team<>("Dragon");
        schoolarTeam1.addNewParticipant(schoolar1);
        schoolarTeam1.addNewParticipant(schoolar2);

        Team<Schoolar> schoolarTeam2 = new Team<>("Rozumnyky");
        schoolarTeam2.addNewParticipant(schoolar3);
        schoolarTeam2.addNewParticipant(schoolar4);

        System.out.println("\n=== Creating Student Teams ===");
        Team<Student> studentTeam1 = new Team<>("Vpered");
        studentTeam1.addNewParticipant(student1);
        studentTeam1.addNewParticipant(student2);

        Team<Student> studentTeam2 = new Team<>("Intellect");
        studentTeam2.addNewParticipant(student3);
        studentTeam2.addNewParticipant(student4);

        System.out.println("\n=== Creating Employee Teams ===");
        Team<Employee> employeeTeam1 = new Team<>("Robotyagi");
        employeeTeam1.addNewParticipant(employee1);
        employeeTeam1.addNewParticipant(employee2);

        Team<Employee> employeeTeam2 = new Team<>("Professionals");
        employeeTeam2.addNewParticipant(employee3);
        employeeTeam2.addNewParticipant(employee4);

        System.out.println("\n=== Games ===");
        System.out.println("Schoolar teams playing:");
        schoolarTeam1.playWith(schoolarTeam2);

        System.out.println("\nStudent teams playing:");
        studentTeam1.playWith(studentTeam2);

        System.out.println("\nEmployee teams playing:");
        employeeTeam1.playWith(employeeTeam2);

        System.out.println("\n=== Demonstration of constraints (commented in code)
===");
        System.out.println("- Cannot add participant from another league to
team");
        System.out.println("- Teams of different leagues cannot play with each
other");
        System.out.println("- All constraints are checked at compile time thanks
to generics!");

        /*
         * The following lines would cause COMPILE-TIME errors:
         *
         * // Error: Cannot add Student to Team<Schoolar>
         * schoolarTeam1.addNewParticipant(student1);
         *
         * // Error: Cannot add Employee to Team<Student>
         * studentTeam1.addNewParticipant(employee1);
         *
         * // Error: Cannot play Team<Schoolar> vs Team<Student>
```

```
         * schoolarTeam1.playWith(studentTeam1);
         */
    }
}
```

JsonFileOperations.java

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

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.*;
import java.nio.file.Files;
import java.nio.file.Paths;

/**
 * Utility class for file-based JSON serialization and deserialization.
 *
 * <p>This class provides convenient methods to persist Java objects to JSON files
 * and load them back from disk. It serves as a high-level wrapper around the
 * {@link JsonParser} class, handling file I/O operations automatically.</p>
 *
 * <h2>Key Features:</h2>
 * <ul>
 *   <li><b>Object to File:</b> Serialize objects to JSON files</li>
 *   <li><b>File to Object:</b> Deserialize JSON files to typed Java objects</li>
 *   <li><b>Automatic Conversion:</b> Uses {@link JsonParser} for JSON
conversion</li>
 *   <li><b>Error Handling:</b> Comprehensive exception handling and logging</li>
 *   <li><b>File I/O:</b> Uses modern NIO methods (Files class) for
efficiency</li>
 * </ul>
 *
 * <h2>Supported Types:</h2>
 * <p>Any type supported by {@link JsonParser} can be saved and loaded:
 * <ul>
 *   <li>All Participant types: Schoolar, Student, Employee</li>
 *   <li>Team classes with generic type parameters</li>
 *   <li>Custom objects with Gson support</li>
 * </ul>
 * </p>
 *
 * <h2>Usage Examples:</h2>
 * <pre>
 * // Save a student to a JSON file
 * Student student = new Student("Alice", 20);
 * JsonFileOperations.saveToJsonFile(student, "student.json");
 * // Creates file with JSON content:
 * // {
 * //   "name": "Alice",
 * //   "age": 20,
 * //   "tempInfo": "Temporary data for Alice"
 * // }
 *
 * // Load student back from file
 * Student loadedStudent = JsonFileOperations.loadFromJsonFile(
 *     "student.json", Student.class
 * );
 * System.out.println(loadedStudent.getName()); // Output: Alice
 *
 * // Save a team to a file
 * Team<Student> team = new Team<>("Eagles");
 * team.addNewParticipant(new Student("Bob", 21));
```

```
 * JsonFileOperations.saveToJsonFile(team, "team.json");
 *
 * // Load team back from file
 * Team<Student> loadedTeam = JsonFileOperations.loadFromJsonFile(
 *     "team.json", Team.class
 * );
 * System.out.println(loadedTeam.getName()); // Output: Eagles
 * </pre>
 *
 * <h2>File Encoding:</h2>
 * <p>All files are read and written using UTF-8 encoding by default.</p>
 *
 * <h2>Error Handling:</h2>
 * <p>Both methods wrap {@link IOException} in {@link RuntimeException} to provide
 * an unchecked exception that can be handled at a higher level. All errors are
 * logged with full details before being thrown.</p>
 *
 * <h2>Performance Notes:</h2>
 * <ul>
 *   <li>Uses {@link java.nio.file.Files} for efficient file I/O</li>
 *   <li>Suitable for small to medium-sized files</li>
 *   <li>For very large files, consider streaming approaches</li>
 * </ul>
 *
 * @author ZTU Student
 * @version 1.0
 * @since 2024
 *
 * @see JsonParser
 * @see Participant
 * @see Team
 * @see java.nio.file.Files
 */
public class JsonFileOperations {

    /**
     * Logger instance for this class.
     * Used for logging file save/load operations and errors.
     */
    private static final Logger logger =
LoggerFactory.getLogger(JsonFileOperations.class);

    /**
     * Saves an object as JSON to a file on disk.
     *
     * <p>This method:
     * <ul>
     *   <li>Converts the object to JSON using {@link
JsonParser#toJson(Object)}</li>
     *   <li>Writes the JSON string to a file at the specified path</li>
     *   <li>Creates the file if it doesn't exist</li>
     *   <li>Overwrites the file if it already exists</li>
     *   <li>Logs the operation at info and debug levels</li>
     * </ul>
     * </p>
     *
     * <h2>Example:</h2>
     * <pre>
     * Student student = new Student("Charlie", 22);
     * JsonFileOperations.saveToJsonFile(student, "data/students/charlie.json");
     * </pre>
     *
     * <h2>Exception Behavior:</h2>
```

```
     * <p>If the save operation fails (e.g., file permissions, parent directory
doesn't exist),
     * an error is logged and a {@link RuntimeException} is thrown.</p>
     *
     * @param object the object to save to a JSON file (must not be null)
     * @param filePath the file path where the JSON will be saved (must not be
null)
     *                 Can be relative (e.g., "file.json") or absolute path
     * @throws RuntimeException if an I/O error occurs during file writing
     * @throws NullPointerException if object or filePath is null
     *
     * @see #loadFromJsonFile(String, Class)
     * @see JsonParser#toJson(Object)
     * @see java.nio.file.Files#write(java.nio.file.Path, byte[])
     */
    public static void saveToJsonFile(Object object, String filePath) {
        logger.info("Saving object to JSON file: {}", filePath);
        try {
            String json = JsonParser.toJson(object);

            Files.write(Paths.get(filePath), json.getBytes());

            logger.info("Object successfully saved to {}", filePath);
            System.out.println("Saved to: " + filePath);
        } catch (IOException e) {
            logger.error("Error saving to file: {}", filePath, e);
            throw new RuntimeException("Failed to save JSON to file", e);
        }
    }

    /**
     * Loads an object from a JSON file on disk.
     *
     * <p>This method:
     * <ul>
     *   <li>Reads the JSON content from the specified file</li>
     *   <li>Deserializes it to the specified class using {@link
JsonParser#fromJson(String, Class)}</li>
     *   <li>Returns the deserialized object</li>
     *   <li>Logs the operation at info and debug levels</li>
     * </ul>
     * </p>
     *
     * <h2>Example:</h2>
     * <pre>
     * Student loadedStudent = JsonFileOperations.loadFromJsonFile(
     *     "data/students/charlie.json", Student.class
     * );
     * System.out.println(loadedStudent.getName());
     * </pre>
     *
     * <h2>Type Parameter:</h2>
     * <p>The {@code clazz} parameter determines the type of object returned.
     * This must match the JSON structure in the file for proper
deserialization.</p>
     *
     * <h2>Exception Behavior:</h2>
     * <p>If the load operation fails (e.g., file not found, invalid JSON),
     * an error is logged and a {@link RuntimeException} is thrown.</p>
     *
     * @param filePath the file path to load from (must not be null)
     *                 Can be relative (e.g., "file.json") or absolute path
     * @param clazz the {@link Class} to deserialize the JSON into
```

```
     * @param <T> the type parameter matching the clazz parameter
     * @return a new instance of class {@code T} populated from the file's JSON
data
     * @throws RuntimeException if an I/O error occurs during file reading
     *                          or if JSON deserialization fails
     * @throws NullPointerException if filePath or clazz is null
     *
     * @see #saveToJsonFile(Object, String)
     * @see JsonParser#fromJson(String, Class)
     * @see java.nio.file.Files#readString(java.nio.file.Path)
     */
    public static <T> T loadFromJsonFile(String filePath, Class<T> clazz) {
        logger.info("Loading object from JSON file: {}", filePath);
        try {
            String json = Files.readString(Paths.get(filePath));

            T object = JsonParser.fromJson(json, clazz);

            logger.info("Object successfully loaded from {}", filePath);
            return object;
        } catch (IOException e) {
            logger.error("Error loading from file: {}", filePath, e);
            throw new RuntimeException("Failed to load JSON from file", e);
        }
    }
}
```

JsonParser.java

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

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * Utility class for JSON parsing and serialization using Google's Gson library.
 *
 * <p>This class provides convenient methods to convert Java objects to JSON
format
 * and deserialize JSON strings back to Java objects. It uses Gson with pretty-
printing
 * enabled for human-readable JSON output.</p>
 *
 * <h2>Key Features:</h2>
 * <ul>
 *   <li><b>Object to JSON:</b> Convert any Java object to formatted JSON
string</li>
 *   <li><b>JSON to Object:</b> Deserialize JSON strings back to typed Java
objects</li>
 *   <li><b>Pretty Printing:</b> JSON output is formatted for readability</li>
 *   <li><b>Logging:</b> Full SLF4J logging support for debugging</li>
 *   <li><b>Error Handling:</b> Comprehensive exception handling and logging</li>
 * </ul>
 *
 * <h2>Supported Types:</h2>
 * <p>This utility can serialize/deserialize:
 * <ul>
 *   <li>All Participant types: Schoolar, Student, Employee</li>
 *   <li>Team classes with generic type parameters</li>
 *   <li>Custom objects with Gson support</li>
 *   <li>Collections and arrays of supported types</li>
 * </ul>
```

```
 * </p>
 *
 * <h2>Usage Examples:</h2>
 * <pre>
 * // Convert object to JSON
 * Student student = new Student("John", 20);
 * String json = JsonParser.toJson(student);
 * System.out.println(json);
 * // Output:
 * // {
 * //   "name": "John",
 * //   "age": 20,
 * //   "tempInfo": "Temporary data for John"
 * // }
 *
 * // Convert JSON back to object
 * String jsonString = "{\"name\":\"Jane\",\"age\":21,\"tempInfo\":\"Temp\"}";
 * Student deserializedStudent = JsonParser.fromJson(jsonString, Student.class);
 * System.out.println(deserializedStudent.getName()); // Output: Jane
 *
 * // Convert team to JSON
 * Team<Student> team = new Team<>("Eagles");
 * team.addNewParticipant(student);
 * String teamJson = JsonParser.toJson(team);
 *
 * // Convert back to team
 * Team<Student> teamDeserialized = JsonParser.fromJson(teamJson, Team.class);
 * </pre>
 *
 * <h2>Exception Handling:</h2>
 * <p>All methods wrap checked exceptions in {@link RuntimeException} and log
errors.
 * This simplifies error handling for the caller while maintaining detailed
logging.</p>
 *
 * <h2>Implementation Notes:</h2>
 * <p>
 * <ul>
 *   <li>Gson instance is created once and reused for performance</li>
 *   <li>Pretty printing is enabled for better readability</li>
 *   <li>All conversions are logged at debug level (with class names)</li>
 *   <li>Successful conversions are logged at trace level</li>
 *   <li>Errors are logged at error level with full exception details</li>
 * </ul>
 * </p>
 *
 * @author ZTU Student
 * @version 1.0
 * @since 2024
 *
 * @see com.google.gson.Gson
 * @see Participant
 * @see Team
 * @see JsonFileOperations
 */
public class JsonParser {

    /**
     * Logger instance for this class.
     * Used for logging JSON conversion operations and errors.
     */
    private static final Logger logger =
LoggerFactory.getLogger(JsonParser.class);
```

```
    /**
     * Gson instance configured with pretty printing.
     * <p>This is a class-level constant that is created once and reused
     * for all JSON operations. Pretty printing is enabled to produce
     * human-readable JSON output.</p>
     */
    private static final Gson gson = new GsonBuilder()
            .setPrettyPrinting()
            .create();

    /**
     * Converts a Java object to a JSON string with pretty printing.
     *
     * <p>This method serializes any object to JSON format. The resulting JSON is
     * formatted with indentation for readability. All fields visible to Gson
     * are included in the output.</p>
     *
     * <h2>Example:</h2>
     * <pre>
     * Student student = new Student("Alice", 22);
     * String json = JsonParser.toJson(student);
     * // json contains formatted JSON representation
     * </pre>
     *
     * @param object the object to convert to JSON (must not be null)
     * @return a formatted JSON string representation of the object
     * @throws RuntimeException if JSON conversion fails (with underlying
exception logged)
     * @throws NullPointerException if object is null (from Gson)
     *
     * @see #fromJson(String, Class)
     * @see com.google.gson.Gson#toJson(Object)
     */
    public static String toJson(Object object) {
        logger.debug("Converting object to JSON: {}",
object.getClass().getSimpleName());
        try {
            String json = gson.toJson(object);
            logger.trace("JSON conversion successful");
            return json;
        } catch (Exception e) {
            logger.error("Error converting object to JSON", e);
            throw new RuntimeException("JSON conversion failed", e);
        }
    }

    /**
     * Converts a JSON string to a typed Java object.
     *
     * <p>This method deserializes JSON data into a Java object of the specified
class.
     * The JSON structure should match the target class structure for successful
     * deserialization.</p>
     *
     * <h2>Example:</h2>
     * <pre>
     * String json = "{\"name\":\"Bob\",\"age\":23}";
     * Student student = JsonParser.fromJson(json, Student.class);
     * System.out.println(student.getName()); // Output: Bob
     * </pre>
     *
     * <h2>Important Notes:</h2>
```

```
     * <ul>
     *   <li>The target class should have appropriate constructors (Gson uses
reflection)</li>
     *   <li>JSON field names should match class field names</li>
     *   <li>Missing fields in JSON will use default values</li>
     *   <li>Extra fields in JSON will be ignored</li>
     * </ul>
     *
     * @param json the JSON string to deserialize (must be valid JSON)
     * @param clazz the {@link Class} to deserialize into
     * @param <T> the type parameter matching the clazz parameter
     * @return a new instance of class {@code T} populated from the JSON data
     * @throws RuntimeException if JSON deserialization fails (with underlying
exception logged)
     * @throws NullPointerException if json or clazz is null
     *
     * @see #toJson(Object)
     * @see com.google.gson.Gson#fromJson(String, Class)
     */
    public static <T> T fromJson(String json, Class<T> clazz) {
        logger.debug("Converting JSON to object of type: {}",
clazz.getSimpleName());
        try {
            T object = gson.fromJson(json, clazz);
            logger.trace("JSON deserialization successful");
            return object;
        } catch (Exception e) {
            logger.error("Error converting JSON to object", e);
            throw new RuntimeException("JSON deserialization failed", e);
        }
    }
}
```

Participant.java

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

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.Serializable;
import java.util.Objects;

/**
 * Abstract base class representing a participant in the game system.
 *
 * <p>This class serves as the foundation for all participant types in the game,
 * implementing essential interfaces for cloning, comparison, and serialization.
 * All participant types (Schoolar, Student, Employee) must extend this class.</p>
 *
 * <h2>Key Features:</h2>
 * <ul>
 *   <li><b>Serializable:</b> Participants can be serialized to files for
persistence</li>
 *   <li><b>Cloneable:</b> Deep copy support for creating participant
duplicates</li>
 *   <li><b>Comparable:</b> Participants can be sorted alphabetically by name</li>
 *   <li><b>Logging:</b> Full SLF4J logging support for debugging</li>
 * </ul>
 *
 * <h2>Field Details:</h2>
 * <ul>
 *   <li><b>name:</b> Participant's full name</li>
 *   <li><b>age:</b> Participant's age (must be non-negative)</li>
```

```
 *   <li><b>tempInfo:</b> Transient field that holds temporary data (not
serialized)</li>
 * </ul>
 *
 * <h2>Usage Example:</h2>
 * <pre>
 * // Note: Cannot instantiate abstract class directly
 * Student student = new Student("John Doe", 20);
 * String name = student.getName();
 * int age = student.getAge();
 * student.setAge(21);
 *
 * // Cloning
 * Student clone = (Student) student.clone();
 *
 * // Comparison
 * int comparison = student.compareTo(new Student("Jane", 19));
 * </pre>
 *
 * @author ZTU Student
 * @version 1.0
 * @since 2024
 * @see Schoolar
 * @see Student
 * @see Employee
 * @see Team
 */
public abstract class Participant implements Cloneable, Comparable<Participant>,
Serializable {

    /**
     * Serial version UID for serialization compatibility.
     * Used to maintain compatibility when serialized classes are updated.
     */
    private static final long serialVersionUID = 1234567890123456789L;

    /**
     * Logger instance for this class.
     * Used for logging participant lifecycle events and state changes.
     */
    private static final Logger logger =
LoggerFactory.getLogger(Participant.class);

    /**
     * The full name of the participant.
     * This field is serialized and persists across serialization/deserialization
cycles.
     */
    private String name;

    /**
     * The age of the participant in years.
     * This field is serialized and persists across serialization/deserialization
cycles.
     * Must be a non-negative integer.
     */
    private int age;

    /**
     * Temporary information about the participant.
     * <p>This field is marked as {@code transient} and will NOT be serialized.
     * After deserialization, this field will be null. This is useful for storing
     * runtime-only information that should not be persisted.</p>
```

```
     */
    private transient String tempInfo;

    /**
     * Constructs a new Participant with the specified name and age.
     *
     * <p>This constructor initializes all fields and sets up temporary
information.
     * The {@code tempInfo} field is automatically populated with a descriptive
message.</p>
     *
     * @param name the name of the participant (cannot be null)
     * @param age the age of the participant (should be non-negative)
     * @throws NullPointerException if name is null
     *
     * @see #tempInfo
     */
    public Participant(String name, int age) {
        logger.debug("Creating participant: name={}, age={}", name, age);
        this.name = name;
        this.age = age;
        this.tempInfo = "Temporary data for " + name;
        logger.trace("Participant created successfully with tempInfo: {}",
tempInfo);
    }

    /**
     * Returns the name of the participant.
     *
     * @return the participant's name as a {@code String}
     *
     * @see #setName(String)
     */
    public String getName() {
        return name;
    }

    /**
     * Returns the age of the participant.
     *
     * @return the participant's age in years as an {@code int}
     *
     * @see #setAge(int)
     */
    public int getAge() {
        return age;
    }

    /**
     * Sets the name of the participant.
     *
     * <p>This method updates the participant's name and logs the change.</p>
     *
     * @param name the new name for the participant (should not be null)
     *
     * @see #getName()
     */
    public void setName(String name) {
        logger.debug("Changing name from '{}' to '{}'", this.name, name);
        this.name = name;
    }

    /**
```

```
     * Sets the age of the participant.
     *
     * <p>This method validates that the age is non-negative before setting it.
     * If a negative age is provided, an exception is thrown and logged.</p>
     *
     * @param age the new age for the participant in years
     * @throws IllegalArgumentException if age is negative (less than 0)
     *
     * @see #getAge()
     */
    public void setAge(int age) {
        if (age < 0) {
            logger.warn("Attempt to set negative age: {}", age);
            throw new IllegalArgumentException("Age cannot be negative");
        }
        logger.debug("Changing age from {} to {}", this.age, age);
        this.age = age;
    }

    /**
     * Returns the temporary information about the participant.
     *
     * <p><b>Important Note:</b> This field is marked as {@code transient} and
will be
     * {@code null} after deserialization. This is by design - temporary data is
not
     * meant to be persisted.</p>
     *
     * @return the temporary information string, or {@code null} if not set or
after deserialization
     *
     * @see #setTempInfo(String)
     */
    public String getTempInfo() {
        return tempInfo;
    }

    /**
     * Sets the temporary information for the participant.
     *
     * <p>Note that this field is transient and will not be serialized.
     * This is useful for storing runtime-only information.</p>
     *
     * @param tempInfo the temporary information to set (can be null)
     *
     * @see #getTempInfo()
     */
    public void setTempInfo(String tempInfo) {
        this.tempInfo = tempInfo;
    }

    /**
     * Creates and returns a deep copy (clone) of this participant.
     *
     * <p>This method uses the Object clone mechanism to create a shallow copy of
the
     * participant. Since {@code String} and {@code int} are immutable, this
effectively
     * creates a deep copy.</p>
     *
     * @return a clone of this participant with identical field values
     * @throws RuntimeException if cloning is not supported by the subclass
     *
```

```
     * @see java.lang.Object#clone()
     */
    @Override
    public Participant clone() {
        try {
            logger.trace("Cloning participant: {}", name);
            Participant cloned = (Participant) super.clone();
            logger.trace("Participant cloned successfully");
            return cloned;
        } catch (CloneNotSupportedException e) {
            logger.error("Clone not supported for participant: {}", name, e);
            throw new RuntimeException("Clone not supported", e);
        }
    }

    /**
     * Indicates whether some other object is "equal to" this one.
     *
     * <p>Two participants are considered equal if and only if:
     * <ul>
     *   <li>They are the same object in memory, OR</li>
     *   <li>They are both Participant instances with identical name AND age</li>
     * </ul>
     * Note: The {@code tempInfo} field is NOT considered in equality
comparison.</p>
     *
     * @param o the reference object with which to compare
     * @return {@code true} if this object is equal to the argument; {@code false}
otherwise
     *
     * @see #hashCode()
     */
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Participant that = (Participant) o;
        return age == that.age && Objects.equals(name, that.name);
    }

    /**
     * Returns a hash code value for the participant.
     *
     * <p>The hash code is calculated based on the {@code name} and {@code age}
fields.
     * Participants with equal name and age will have the same hash code.</p>
     *
     * @return a hash code value for this participant
     *
     * @see #equals(Object)
     * @see java.util.Objects#hash(Object...)
     */
    @Override
    public int hashCode() {
        return Objects.hash(name, age);
    }

    /**
     * Returns a string representation of the participant.
     *
     * <p>The string includes the participant's name, age, and temporary info.
     * Format: {@code Participant{name='...', age=..., tempInfo='...'}}</p>
     *
```

```
     * @return a string representation containing participant's data
     */
    @Override
    public String toString() {
        return "Participant{" +
                "name='" + name + '\'' +
                ", age=" + age +
                ", tempInfo='" + tempInfo + '\'' +
                '}';
    }

    /**
     * Compares this participant with the specified participant for order.
     *
     * <p>Participants are ordered alphabetically by their names using natural
     * {@code String} comparison. This allows participants to be sorted in
     * lexicographic order.</p>
     *
     * @param other the participant to be compared (must not be null)
     * @return a negative integer if this participant's name is less than the
other's;
     *         zero if the names are equal;
     *         a positive integer if this participant's name is greater than the
other's
     *
     * @throws NullPointerException if the other participant is null
     *
     * @see java.lang.String#compareTo(String)
     */
    @Override
    public int compareTo(Participant other) {
        logger.trace("Comparing {} with {}", this.name, other.name);
        return this.name.compareTo(other.name);
    }
}
```

Schoolar.java

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

/**
 * Represents a school-age student participant in the game system.
 *
 * <p>This class extends the abstract {@link Participant} class and specializes it
for
 * school-age participants, typically ages 6-17. Schoolar (School) participants
form
 * teams of their own type and compete with other schoolar teams.</p>
 *
 * <h2>Characteristics:</h2>
 * <ul>
 *   <li><b>Age Range:</b> Typically 6-17 years old</li>
 *   <li><b>Team Composition:</b> Teams can only contain other Schoolar
participants</li>
 *   <li><b>Serialization:</b> Fully serializable with custom
serialVersionUID</li>
 * </ul>
 *
 * <h2>Type Safety:</h2>
 * <p>Through Java Generics in the {@link Team} class, a {@code Team<Schoolar>}
can only
 * contain Schoolar participants. This prevents mixing different participant types
in
 * the same team and is enforced at compile time.</p>
```

```
 *
 * <h2>Usage Example:</h2>
 * <pre>
 * // Create individual schoolar participants
 * Schoolar ivan = new Schoolar("Ivan", 13);
 * Schoolar mariya = new Schoolar("Mariya", 15);
 *
 * // Create a team for schoolars
 * Team&lt;Schoolar&gt; dragonTeam = new Team&lt;&gt;("Dragon");
 *
 * // Add schoolars to the team
 * dragonTeam.addNewParticipant(ivan);
 * dragonTeam.addNewParticipant(mariya);
 *
 * // Create another team and play
 * Team&lt;Schoolar&gt; wizardTeam = new Team&lt;&gt;("Wizards");
 * dragonTeam.playWith(wizardTeam);
 * </pre>
 *
 * <h2>Inherited Features:</h2>
 * <p>Schoolar inherits all functionality from Participant including:
 * <ul>
 *   <li>Name and age management ({@link #getName()}, {@link #getAge()})</li>
 *   <li>Cloning support ({@link #clone()})</li>
 *   <li>Serialization support</li>
 *   <li>Natural ordering by name ({@link #compareTo(Participant)})</li>
 *   <li>Equality and hashing ({@link #equals(Object)}, {@link #hashCode()})</li>
 * </ul>
 * </p>
 *
 * @author ZTU Student
 * @version 1.0
 * @since 2024
 *
 * @see Participant
 * @see Student
 * @see Employee
 * @see Team
 * @see Game
 */
public class Schoolar extends Participant {

    /**
     * Serial version UID for serialization.
     * <p>This unique identifier ensures that Schoolar objects can be safely
     * serialized and deserialized across different versions of the class.</p>
     */
    private static final long serialVersionUID = 1111111111111111111L;

    /**
     * Constructs a new Schoolar participant with the specified name and age.
     *
     * <p>This constructor delegates to the parent {@link Participant} class
     * constructor, which initializes the name, age, and temporary
information.</p>
     *
     * @param name the name of the schoolar (e.g., "Ivan", "Mariya")
     * @param age the age of the schoolar in years (typically 6-17)
     *
     * @throws NullPointerException if name is null
     *
     * @see Participant#Participant(String, int)
     */
```

```
    public Schoolar(String name, int age) {
        super(name, age);
    }

    /**
     * Returns a string representation of this Schoolar participant.
     *
     * <p>The string includes the schoolar's name, age, and temporary information.
     * Format: {@code Schoolar{name='...', age=..., tempInfo='...'}}</p>
     *
     * <p>This method overrides the parent {@link Participant#toString()} to
provide
     * a Schoolar-specific string representation.</p>
     *
     * @return a string representation containing {@code Schoolar{name, age,
tempInfo}}
     */
    @Override
    public String toString() {
        return "Schoolar{" +
                "name='" + getName() + '\'' +
                ", age=" + getAge() +
                ", tempInfo='" + getTempInfo() + '\'' +
                '}';
    }
}
```

Student.java

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

/**
 * Represents a university/college student participant in the game system.
 *
 * <p>This class extends the abstract {@link Participant} class and specializes it
for
 * university and college students, typically ages 18-25. Student participants
form
 * their own teams and compete with other student teams.</p>
 *
 * <h2>Characteristics:</h2>
 * <ul>
 *   <li><b>Age Range:</b> Typically 18-25 years old</li>
 *   <li><b>Team Composition:</b> Teams can only contain other Student
participants</li>
 *   <li><b>Serialization:</b> Fully serializable with custom
serialVersionUID</li>
 * </ul>
 *
 * <h2>Type Safety:</h2>
 * <p>Through Java Generics in the {@link Team} class, a {@code Team<Student>} can
only
 * contain Student participants. This prevents mixing different participant types
in
 * the same team and is enforced at compile time. For example, you cannot add a
Schoolar
 * or Employee to a Student team.</p>
 *
 * <h2>Usage Example:</h2>
 * <pre>
 * // Create individual student participants
 * Student mykola = new Student("Mykola", 20);
 * Student viktoria = new Student("Viktoria", 21);
 *
```

```
 * // Create a team for students
 * Team&lt;Student&gt; vperedTeam = new Team&lt;&gt;("Vpered");
 *
 * // Add students to the team
 * vperedTeam.addNewParticipant(mykola);
 * vperedTeam.addNewParticipant(viktoria);
 *
 * // Create another team and play a game
 * Team&lt;Student&gt; intellectTeam = new Team&lt;&gt;("Intellect");
 * vperedTeam.playWith(intellectTeam);
 * </pre>
 *
 * <h2>Inherited Features:</h2>
 * <p>Student inherits all functionality from Participant including:
 * <ul>
 *   <li>Name and age management ({@link #getName()}, {@link #getAge()})</li>
 *   <li>Cloning support ({@link #clone()})</li>
 *   <li>Serialization support</li>
 *   <li>Natural ordering by name ({@link #compareTo(Participant)})</li>
 *   <li>Equality and hashing ({@link #equals(Object)}, {@link #hashCode()})</li>
 * </ul>
 * </p>
 *
 * <h2>Differences from Other Participant Types:</h2>
 * <p>
 * While Schoolar represents younger students and Employee represents working
professionals,
 * Student represents university/college students, a distinct group with their own
 * developmental characteristics and competing interests.
 * </p>
 *
 * @author ZTU Student
 * @version 1.0
 * @since 2024
 *
 * @see Participant
 * @see Schoolar
 * @see Employee
 * @see Team
 * @see Game
 */
public class Student extends Participant {

    /**
     * Serial version UID for serialization.
     * <p>This unique identifier ensures that Student objects can be safely
     * serialized and deserialized across different versions of the class.</p>
     */
    private static final long serialVersionUID = 2222222222222222222L;

    /**
     * Constructs a new Student participant with the specified name and age.
     *
     * <p>This constructor delegates to the parent {@link Participant} class
     * constructor, which initializes the name, age, and temporary
information.</p>
     *
     * @param name the name of the student (e.g., "Mykola", "Viktoria")
     * @param age the age of the student in years (typically 18-25)
     *
     * @throws NullPointerException if name is null
     *
     * @see Participant#Participant(String, int)
```

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

    /**
     * Returns a string representation of this Student participant.
     *
     * <p>The string includes the student's name, age, and temporary information.
     * Format: {@code Student{name='...', age=..., tempInfo='...'}}</p>
     *
     * <p>This method overrides the parent {@link Participant#toString()} to
provide
     * a Student-specific string representation.</p>
     *
     * @return a string representation containing {@code Student{name, age,
tempInfo}}
     */
    @Override
    public String toString() {
        return "Student{" +
                "name='" + getName() + '\'' +
                ", age=" + getAge() +
                ", tempInfo='" + getTempInfo() + '\'' +
                '}';
    }
}
```

Team.java

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

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
import java.util.Objects;
import java.util.Random;

/**
 * Represents a team of participants in the game system.
 *
 * <p>This generic class provides a type-safe container for participants using
Java Generics.
 * A team can hold multiple participants of the same type (e.g., all Students, all
Employees),
 * ensuring compile-time type safety. Teams can compete against other teams of the
same type.</p>
 *
 * <h2>Generic Type Safety:</h2>
 * <p>The Team class is generic with type parameter {@code <T extends
Participant>}.
 * This ensures:
 * <ul>
 *   <li>A {@code Team<Schoolar>} can only contain Schoolar participants</li>
 *   <li>A {@code Team<Student>} can only contain Student participants</li>
 *   <li>A {@code Team<Employee>} can only contain Employee participants</li>
 *   <li>These constraints are enforced at compile time, not runtime</li>
 * </ul>
 * </p>
 *
 * <h2>Key Features:</h2>
```

```
 * <ul>
 *   <li><b>Serializable:</b> Teams can be serialized and persisted</li>
 *   <li><b>Transient Fields:</b> Rating and wins are not serialized</li>
 *   <li><b>Deep Cloning:</b> Create independent copies of teams</li>
 *   <li><b>Game Simulation:</b> Teams can play against each other</li>
 * </ul>
 *
 * <h2>Transient Fields:</h2>
 * <p>The {@code rating} and {@code wins} fields are marked as {@code transient}
and
 * will not be serialized. This means these game statistics are reset when a team
is
 * deserialized. This is intentional for separating persistent data (participants)
 * from runtime data (statistics).</p>
 *
 * <h2>Usage Example:</h2>
 * <pre>
 * // Create participants
 * Student student1 = new Student("John", 20);
 * Student student2 = new Student("Jane", 21);
 *
 * // Create a team
 * Team&lt;Student&gt; team = new Team&lt;&gt;("Eagles");
 *
 * // Add participants to the team
 * team.addNewParticipant(student1);
 * team.addNewParticipant(student2);
 *
 * // Create another team
 * Team&lt;Student&gt; otherTeam = new Team&lt;&gt;("Hawks");
 * otherTeam.addNewParticipant(new Student("Mike", 22));
 *
 * // Teams play against each other
 * team.playWith(otherTeam);
 *
 * // Clone a team
 * Team&lt;Student&gt; teamClone = Team.deepClone(team);
 * </pre>
 *
 * <h2>Constructor Details:</h2>
 * <ul>
 *   <li><b>Team(String):</b> Creates a new team with a name</li>
 *   <li><b>Team(Team<T>):</b> Creates a deep copy of another team</li>
 * </ul>
 *
 * @param <T> the type of participants in this team, must extend {@link
Participant}
 *
 * @author ZTU Student
 * @version 1.0
 * @since 2024
 *
 * @see Participant
 * @see Schoolar
 * @see Student
 * @see Employee
 * @see Game
 */
public class Team<T extends Participant> implements Serializable {

    /**
     * Serial version UID for serialization.
     * <p>This unique identifier ensures that Team objects can be safely
```

```
     * serialized and deserialized across different versions of the class.</p>
     */
    private static final long serialVersionUID = 9876543210987643 21L;

    /**
     * Logger instance for this class.
     * Used for logging team lifecycle events, participant additions, and game
results.
     */
    private static final Logger logger = LoggerFactory.getLogger(Team.class);

    /**
     * The name of the team.
     * <p>This field is serialized and persists across
serialization/deserialization cycles.
     * Examples: "Dragons", "Vpered", "Robotyagi"</p>
     */
    private String name;

    /**
     * The list of participants in this team.
     * <p>This field is serialized and persists across
serialization/deserialization cycles.
     * All participants must be of type {@code T}, enforced by Java Generics.</p>
     */
    private List<T> participants = new ArrayList<>();

    /**
     * The current rating of the team.
     * <p>This field is marked as {@code transient} and will NOT be serialized.
     * Initial rating is 1000. The rating is updated through gameplay.</p>
     */
    private transient int rating;

    /**
     * The number of games won by this team.
     * <p>This field is marked as {@code transient} and will NOT be serialized.
     * Initial wins count is 0. The wins counter is incremented with each
victory.</p>
     */
    private transient int wins;

    /**
     * Constructs a new Team with the specified name.
     *
     * <p>This constructor initializes the team with:
     * <ul>
     *   <li>The provided team name</li>
     *   <li>An empty list of participants</li>
     *   <li>An initial rating of 1000</li>
     *   <li>An initial wins count of 0</li>
     * </ul>
     * </p>
     *
     * @param name the name of the team (e.g., "Dragons", "Vpered")
     *
     * @see #getName()
     */
    public Team(String name) {
        logger.info("Creating team: {}", name);
        this.name = name;
        this.rating = 1000;
        this.wins = 0;
```

```
        logger.debug("Team '{}' created with initial rating: {}, wins: {}", name,
rating, wins);
    }

    /**
     * Constructs a new Team as a deep copy of another team.
     *
     * <p>This copy constructor creates a completely independent team with:
     * <ul>
     *   <li>The same name as the original</li>
     *   <li>Deep copies of all participants (clones)</li>
     *   <li>The same rating as the original</li>
     *   <li>The same wins count as the original</li>
     * </ul>
     * The original team and the copy are completely independent. Modifying one
     * does not affect the other.</p>
     *
     * @param other the team to copy (must not be null)
     *
     * @throws NullPointerException if other is null
     *
     * @see #deepClone(Team)
     */
    public Team(Team<T> other) {
        logger.debug("Creating deep copy of team: {}", other.name);
        this.name = other.name;
        this.participants = new ArrayList<>();
        for (T participant : other.participants) {
            this.participants.add((T) participant.clone());
        }
        this.rating = other.rating;
        this.wins = other.wins;
        logger.trace("Team copy created with {} participants",
participants.size());
    }

    /**
     * Creates a deep copy of a team using the copy constructor.
     *
     * <p>This static factory method provides an alternative way to clone a team.
     * It is equivalent to calling {@code new Team<>(original)} but is more
explicit
     * about the intent to create a deep clone.</p>
     *
     * @param original the team to clone (must not be null)
     * @param <T> the type of participants in the team
     * @return a new Team with the same data but independent from the original
     *
     * @throws NullPointerException if original is null
     *
     * @see #Team(Team)
     */
    public static <T extends Participant> Team<T> deepClone(Team<T> original) {
        logger.debug("Deep cloning team using static method");
        return new Team<>(original);
    }

    /**
     * Adds a new participant to the team.
     *
     * <p>This method adds the specified participant to the team's participant
list
     * and logs the action. The participant is added to the end of the list.</p>
```

```
     *
     * @param participant the participant to add (must not be null)
     * @throws IllegalArgumentException if participant is null
     *
     * @see #getParticipants()
     * @see #Team#playWith(Team)
     */
    public void addNewParticipant(T participant) {
        if (participant == null) {
            logger.error("Attempt to add null participant to team '{}'", name);
            throw new IllegalArgumentException("Participant cannot be null");
        }

        logger.info("Adding participant '{}' to team '{}'", participant.getName(),
name);
        participants.add(participant);
        System.out.println("To the team " + name + " was added participant " +
participant.getName());
        logger.debug("Team '{}' now has {} participants", name,
participants.size());
    }

    /**
     * Simulates a game between this team and another team of the same type.
     *
     * <p>This method:
     * <ul>
     *   <li>Randomly selects a winner between the two teams</li>
     *   <li>Increments the wins counter for the winning team</li>
     *   <li>Logs the game result and winner information</li>
     *   <li>Prints the result to the console</li>
     * </ul>
     * The outcome is determined purely by chance using {@link
java.util.Random}.</p>
     *
     * @param team the opposing team to play against (must not be null)
     * @throws IllegalArgumentException if team is null
     *
     * @see #getName()
     * @see #wins
     */
    public void playWith(Team<T> team) {
        if (team == null) {
            logger.error("Attempt to play with null team");
            throw new IllegalArgumentException("Team cannot be null");
        }

        logger.info("Game started: '{}' vs '{}'", this.name, team.name);

        String winnerName;
        Random random = new Random();
        int i = random.nextInt(2);

        if (i == 0) {
            winnerName = this.name;
            this.wins++;
            logger.debug("Team '{}' won! Total wins: {}", this.name, this.wins);
        } else {
            winnerName = team.name;
            team.wins++;
            logger.debug("Team '{}' won! Total wins: {}", team.name, team.wins);
        }
```

```
        System.out.println("The team " + winnerName + " is winner!");
        logger.info("Game finished. Winner: {}", winnerName);
    }

    /**
     * Returns the name of the team.
     *
     * @return the team's name as a {@code String}
     *
     * @see #setName(String)
     */
    public String getName() {
        return name;
    }

    /**
     * Returns the list of all participants in the team.
     *
     * <p>The returned list is the actual list used internally, not a copy.
     * Modifications to the returned list will affect the team.</p>
     *
     * @return a {@code List} of participants of type {@code T}
     *
     * @see #setParticipants(List)
     * @see #addNewParticipant(Participant)
     */
    public List<T> getParticipants() {
        return participants;
    }

    /**
     * Sets the name of the team.
     *
     * <p>This method updates the team's name and logs the change.</p>
     *
     * @param name the new name for the team
     *
     * @see #getName()
     */
    public void setName(String name) {
        logger.debug("Renaming team from '{}' to '{}'", this.name, name);
        this.name = name;
    }

    /**
     * Sets the list of participants for the team.
     *
     * <p>This method replaces the entire participants list with a new one.
     * If the provided list is null, it creates an empty list instead.</p>
     *
     * @param participants the new list of participants (can be null, which
becomes empty list)
     *
     * @see #getParticipants()
     */
    public void setParticipants(List<T> participants) {
        if (participants == null) {
            logger.warn("Attempt to set null participants list");
            this.participants = new ArrayList<>();
        } else {
            logger.debug("Setting participants list with {} members",
participants.size());
            this.participants = participants;
```

```
        }
    }

    /**
     * Returns a string representation of the team.
     *
     * <p>The string includes the team's name and the list of all participants.
     * Format: {@code Team{name='...', participants=[...]}}</p>
     *
     * @return a string representation containing team name and participants
     */
    @Override
    public String toString() {
        return "Team{" +
                "name='" + name + '\'' +
                ", participants=" + participants +
                '}';
    }

    /**
     * Indicates whether some other object is "equal to" this one.
     *
     * <p>Two teams are considered equal if and only if:
     * <ul>
     *   <li>They are the same object in memory, OR</li>
     *   <li>They are both Team instances with identical name AND identical
participants list</li>
     * </ul>
     * Note: The {@code rating} and {@code wins} fields are NOT considered in
equality comparison.</p>
     *
     * @param o the reference object with which to compare
     * @return {@code true} if this object is equal to the argument; {@code false}
otherwise
     *
     * @see #hashCode()
     */
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Team<?> team = (Team<?>) o;
        return Objects.equals(name, team.name) && Objects.equals(participants,
team.participants);
    }

    /**
     * Returns a hash code value for the team.
     *
     * <p>The hash code is calculated based on the {@code name} and {@code
participants} fields.
     * Teams with equal name and identical participants will have the same hash
code.</p>
     *
     * @return a hash code value for this team
     *
     * @see #equals(Object)
     * @see java.util.Objects#hash(Object...)
     */
    @Override
    public int hashCode() {
        return Objects.hash(name, participants);
```

```
    }
}
```

Директорія com.education.ztu

Task_2.java

```
package com.education.ztu;

import com.education.ztu.game.*;

import java.io.*;

public class Task_2 {
    public static void main(String[] args) {
        System.out.println("=== ЗАВДАННЯ 2: Серіалізація та Десеріалізація
===\n");

        Schoolar schoolar = new Schoolar("Ivan", 13);
        Student student = new Student("Mykola", 20);
        Employee employee = new Employee("Andriy", 28);

        Team<Student> studentTeam = new Team<>("Vpered");
        studentTeam.addNewParticipant(student);
        studentTeam.addNewParticipant(new Student("Viktoria", 21));

        System.out.println("\n--- Об'єкти ДО серіалізації ---");
        System.out.println("Schoolar: " + schoolar);
        System.out.println("Student: " + student);
        System.out.println("Employee: " + employee);
        System.out.println("Team: " + studentTeam);
        System.out.println("Student tempInfo (до серіалізації): " +
student.getTempInfo());

        serializeObjects(schoolar, student, employee, studentTeam);

        System.out.println("\n--- Об'єкти ПІСЛЯ десеріалізації ---");
        Schoolar deserializedSchoolar = (Schoolar)
deserializeObject("schoolar.ser");
        Student deserializedStudent = (Student) deserializeObject("student.ser");
        Employee deserializedEmployee = (Employee)
deserializeObject("employee.ser");
        Team<Student> deserializedTeam = (Team<Student>)
deserializeObject("team.ser");

        if (deserializedSchoolar != null) {
            System.out.println("Schoolar: " + deserializedSchoolar);
        }
        if (deserializedStudent != null) {
            System.out.println("Student: " + deserializedStudent);
            System.out.println("Student tempInfo: " +
deserializedStudent.getTempInfo() + " (було виключено з серіалізації!)");
        }
        if (deserializedEmployee != null) {
            System.out.println("Employee: " + deserializedEmployee);
        }
        if (deserializedTeam != null) {
            System.out.println("Team: " + deserializedTeam);
        }

        System.out.println("\n--- Аналіз результатів ---");
        System.out.println("✓ Поля з модифікатором 'transient' НЕ були серіалізо-
вані");
        System.out.println("✓ Після десеріалізації transient поля мають значення
```

```
за замовчуванням:");
        System.out.println("  - tempInfo (String) = null");
        System.out.println("✓ Всі інші поля були успішно серіалізовані та десеріа-
лізовані");
    }

    private static void serializeObjects(Schoolar schoolar, Student student,
                                         Employee employee, Team<Student> team) {
        System.out.println("\n--- Серіалізація об'єктів ---");

        serializeObject(schoolar, "schoolar.ser");
        serializeObject(student, "student.ser");
        serializeObject(employee, "employee.ser");
        serializeObject(team, "team.ser");

        System.out.println("Всі об'єкти успішно серіалізовані!");
    }


    private static void serializeObject(Object obj, String filename) {
        try (ObjectOutputStream oos = new ObjectOutputStream(
                new FileOutputStream(filename))) {
            oos.writeObject(obj);
            System.out.println("✓ Серіалізовано: " + filename);
        } catch (IOException e) {
            System.err.println("Помилка при серіалізації " + filename + ": " +
e.getMessage());
        }
    }


    private static Object deserializeObject(String filename) {
        try (ObjectInputStream ois = new ObjectInputStream(
                new FileInputStream(filename))) {
            Object obj = ois.readObject();
            System.out.println("✓ Десеріалізовано: " + filename);
            return obj;
        } catch (IOException | ClassNotFoundException e) {
            System.err.println("Помилка при десеріалізації " + filename + ": " +
e.getMessage());
            return null;
        }
    }
}
```

Task_3.java

```
package com.education.ztu;

import com.education.ztu.game.*;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class Task_3 {
    private static final Logger logger = LoggerFactory.getLogger(Task_3.class);

    public static void main(String[] args) {
        logger.info("=== ЗАВДАННЯ 3: Демонстрація логування ===");
        logger.trace("Starting application with TRACE level logging");
        logger.trace("JVM Version: {}", System.getProperty("java.version"));
        logger.debug("Creating participants for demonstration");

        try {
```

```
            demonstrateParticipantCreation();
            demonstrateTeamCreation();
            demonstrateGameplay();
            demonstrateErrorHandling();
            logger.info("Application completed successfully");

        } catch (Exception e) {
            logger.error("Critical error occurred during execution", e);
        }

        logger.trace("Application finished");
    }

    private static void demonstrateParticipantCreation() {
        logger.info("--- Demonstration 1: Creating Participants ---");
        logger.debug("Creating schoolar participants");

        Schoolar schoolar1 = new Schoolar("Ivan", 13);
        logger.trace("Schoolar1 created: {}", schoolar1);

        Schoolar schoolar2 = new Schoolar("Mariya", 15);
        logger.trace("Schoolar2 created: {}", schoolar2);

        logger.debug("Creating student participants");
        Student student1 = new Student("Mykola", 20);
        Student student2 = new Student("Viktoria", 21);

        logger.debug("Creating employee participants");
        Employee employee1 = new Employee("Andriy", 28);
        Employee employee2 = new Employee("Oksana", 25);

        logger.info("Successfully created {} participants", 6);
    }

    private static void demonstrateTeamCreation() {
        logger.info("--- Demonstration 2: Creating Teams ---");

        logger.debug("Creating schoolar team");
        Team<Schoolar> schoolarTeam = new Team<>("Dragon");

        Schoolar schoolar1 = new Schoolar("Ivan", 13);
        Schoolar schoolar2 = new Schoolar("Mariya", 15);

        schoolarTeam.addNewParticipant(schoolar1);
        schoolarTeam.addNewParticipant(schoolar2);

        logger.info("Schoolar team '{}' created with {} members",
                schoolarTeam.getName(), schoolarTeam.getParticipants().size());

        logger.debug("Creating student team");
        Team<Student> studentTeam = new Team<>("Vpered");

        Student student1 = new Student("Mykola", 20);
        Student student2 = new Student("Viktoria", 21);

        studentTeam.addNewParticipant(student1);
        studentTeam.addNewParticipant(student2);

        logger.info("Student team '{}' created with {} members",
                studentTeam.getName(), studentTeam.getParticipants().size());
    }

    private static void demonstrateGameplay() {
```

```
        logger.info("--- Demonstration 3: Team Gameplay ---");

        Team<Student> team1 = new Team<>("Alpha");
        Team<Student> team2 = new Team<>("Beta");

        team1.addNewParticipant(new Student("Alex", 22));
        team1.addNewParticipant(new Student("Maria", 21));

        team2.addNewParticipant(new Student("John", 23));
        team2.addNewParticipant(new Student("Kate", 20));

        logger.debug("Both teams are ready for the game");
        logger.info("Starting game between '{}' and '{}'", team1.getName(),
team2.getName());

        team1.playWith(team2);

        logger.info("Game completed successfully");
    }

    private static void demonstrateErrorHandling() {
        logger.info("--- Demonstration 4: Error Handling and Different Log Levels
---");

        logger.warn("Attempting to create participant with unusual age");

        try {
            Schoolar youngSchoolar = new Schoolar("TestChild", 5);
            logger.debug("Young schoolar created: age={}",
youngSchoolar.getAge());

            logger.warn("Attempting to set negative age - this should cause an
error");
            youngSchoolar.setAge(-5);

        } catch (IllegalArgumentException e) {
            logger.error("Failed to set age: {}", e.getMessage());
            logger.debug("Exception details", e);
        }

        logger.warn("Testing null handling in Team");
        try {
            Team<Student> team = new Team<>("TestTeam");
            logger.debug("Attempting to add null participant");
            team.addNewParticipant(null);
        } catch (IllegalArgumentException e) {
            logger.error("Cannot add null participant: {}", e.getMessage());
        }

        logger.info("Error handling demonstration completed");

        logger.error("This would be FATAL level in Log4j - critical system failure
simulation");

        logger.trace("TRACE: Most detailed information for debugging flow");
        logger.debug("DEBUG: Detailed information for diagnosing problems");
        logger.info("INFO: General informational messages");
        logger.warn("WARN: Warning messages about potential problems");
        logger.error("ERROR: Error messages about failures");

        logger.info("All logging levels demonstrated successfully");
    }
}
```

Task_5.java

```java
package com.education.ztu;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.w3c.dom.*;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import java.io.File;

public class Task_5 {
    private static final Logger logger = LoggerFactory.getLogger(Task_5.class);
    private static final String XML_FILE = "teams.xml";
    private static final String OUTPUT_FILE = "teams_modified.xml";

    public static void main(String[] args) {
        logger.info("=== ЗАВДАННЯ 5: XML парсери (DOM) ===");

        try {
            logger.info("Reading XML file: {}", XML_FILE);
            Document document = readXMLFile(XML_FILE);

            logger.info("Parsing and displaying XML content");
            displayXMLContent(document);

            logger.info("Modifying XML document");
            modifyXMLDocument(document);

            logger.info("Saving modified XML to: {}", OUTPUT_FILE);
            saveXMLFile(document, OUTPUT_FILE);

            logger.info("Reading and displaying modified XML");
            Document modifiedDocument = readXMLFile(OUTPUT_FILE);
            displayXMLContent(modifiedDocument);

            logger.info("Task completed successfully!");

        } catch (Exception e) {
            logger.error("Error processing XML", e);
            e.printStackTrace();
        }
    }

    private static Document readXMLFile(String filename) throws Exception {
        logger.debug("Creating DocumentBuilder");
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = factory.newDocumentBuilder();

        logger.debug("Parsing XML file: {}", filename);
        Document document = builder.parse(new File(filename));
        document.getDocumentElement().normalize();

        logger.info("XML file parsed successfully");
        return document;
    }

    private static void displayXMLContent(Document document) {
        System.out.println("\n" + "=".repeat(80));
        System.out.println("XML CONTENT");
        System.out.println("=".repeat(80));
```

```
        Element root = document.getDocumentElement();
        System.out.println("Root element: " + root.getNodeName());

        NodeList teamList = document.getElementsByTagName("team");
        System.out.println("Total teams: " + teamList.getLength());
        System.out.println();

        for (int i = 0; i < teamList.getLength(); i++) {
            Node teamNode = teamList.item(i);

            if (teamNode.getNodeType() == Node.ELEMENT_NODE) {
                Element teamElement = (Element) teamNode;

                String id = teamElement.getAttribute("id");
                String type = teamElement.getAttribute("type");
                String name = getElementTextContent(teamElement, "name");
                String rating = getElementTextContent(teamElement, "rating");
                String wins = getElementTextContent(teamElement, "wins");

                System.out.println("Team #" + id + " [" + type + "]");
                System.out.println("  Name: " + name);
                System.out.println("  Rating: " + rating);
                System.out.println("  Wins: " + wins);

                NodeList participants =
teamElement.getElementsByTagName("participant");
                System.out.println("  Participants (" + participants.getLength() +
"):");

                for (int j = 0; j < participants.getLength(); j++) {
                    Element participant = (Element) participants.item(j);
                    String pName = getElementTextContent(participant, "name");
                    String pAge = getElementTextContent(participant, "age");
                    String pRole = getElementTextContent(participant, "role");

                    System.out.println("    - " + pName + " (Age: " + pAge + ",
Role: " + pRole + ")");
                }
                System.out.println();
            }
        }
        System.out.println("=".repeat(80) + "\n");
    }

    private static void modifyXMLDocument(Document document) {
        logger.debug("Starting XML modification");

        NodeList teamList = document.getElementsByTagName("team");

        for (int i = 0; i < teamList.getLength(); i++) {
            Element teamElement = (Element) teamList.item(i);

            Element ratingElement = (Element)
teamElement.getElementsByTagName("rating").item(0);
            int currentRating = Integer.parseInt(ratingElement.getTextContent());
            int newRating = currentRating + 50;
            ratingElement.setTextContent(String.valueOf(newRating));
            logger.debug("Updated rating for team {} from {} to {}",
                    teamElement.getAttribute("id"), currentRating, newRating);
        }

        Element newTeam = document.createElement("team");
```

```
        newTeam.setAttribute("id", "5");
        newTeam.setAttribute("type", "student");

        Element name = document.createElement("name");
        name.setTextContent("Intellect");
        newTeam.appendChild(name);

        Element rating = document.createElement("rating");
        rating.setTextContent("1000");
        newTeam.appendChild(rating);

        Element wins = document.createElement("wins");
        wins.setTextContent("0");
        newTeam.appendChild(wins);

        Element participants = document.createElement("participants");

        Element participant1 = document.createElement("participant");
        Element pName1 = document.createElement("name");
        pName1.setTextContent("Andriy");
        Element pAge1 = document.createElement("age");
        pAge1.setTextContent("22");
        Element pRole1 = document.createElement("role");
        pRole1.setTextContent("Captain");
        participant1.appendChild(pName1);
        participant1.appendChild(pAge1);
        participant1.appendChild(pRole1);

        Element participant2 = document.createElement("participant");
        Element pName2 = document.createElement("name");
        pName2.setTextContent("Kateryna");
        Element pAge2 = document.createElement("age");
        pAge2.setTextContent("19");
        Element pRole2 = document.createElement("role");
        pRole2.setTextContent("Player");
        participant2.appendChild(pName2);
        participant2.appendChild(pAge2);
        participant2.appendChild(pRole2);

        participants.appendChild(participant1);
        participants.appendChild(participant2);
        newTeam.appendChild(participants);

        document.getDocumentElement().appendChild(newTeam);
        logger.info("Added new team 'Intellect' to XML");

        System.out.println("\n✓ XML document modified:");
        System.out.println("  - Increased all team ratings by 50");
        System.out.println("  - Added new team 'Intellect'\n");
    }

    private static void saveXMLFile(Document document, String filename) throws
Exception {
        logger.debug("Creating Transformer for XML output");
        TransformerFactory transformerFactory = TransformerFactory.newInstance();
        Transformer transformer = transformerFactory.newTransformer();

        transformer.setOutputProperty(OutputKeys.INDENT, "yes");
        transformer.setOutputProperty(OutputKeys.ENCODING, "UTF-8");
        transformer.setOutputProperty("{http://xml.apache.org/xslt}indent-amount",
"4");

        DOMSource source = new DOMSource(document);
```

```
        StreamResult result = new StreamResult(new File(filename));

        logger.debug("Writing XML to file: {}", filename);
        transformer.transform(source, result);

        logger.info("XML file saved successfully: {}", filename);
        System.out.println("✓ XML saved to: " + filename + "\n");
    }

    private static String getElementTextContent(Element parent, String tagName) {
        NodeList nodeList = parent.getElementsByTagName(tagName);
        if (nodeList.getLength() > 0) {
            return nodeList.item(0).getTextContent();
        }
        return "";
    }
}
```

Task_6

```
package com.education.ztu;

import com.education.ztu.game.*;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * Task_6: JSON Parser - Serialization and Deserialization
 * Demonstrates conversion of game entities (Participant types and Teams)
 * to JSON format and back to Java objects using Gson library.
 *
 * @author ZTU Student
 * @version 1.0
 * @since 2024
 */
public class Task_6 {

    private static final Logger logger = LoggerFactory.getLogger(Task_6.class);

    public static void main(String[] args) {
        logger.info("Starting Task_6: JSON Parser");
        System.out.println("=== Task 6: JSON Parser - Java to JSON and JSON to
Java ===\n");

        System.out.println("--- 1. Schoolar to JSON and back ---");
        Schoolar schoolar = new Schoolar("Ivan", 13);
        System.out.println("Original Schoolar: " + schoolar);

        String schoolarJson = JsonParser.toJson(schoolar);
        System.out.println("JSON:\n" + schoolarJson);

        Schoolar schoolarDeserialized = JsonParser.fromJson(schoolarJson,
Schoolar.class);
        System.out.println("Deserialized Schoolar: " + schoolarDeserialized);

        System.out.println("\n--- 2. Student to JSON and back ---");
        Student student = new Student("Mykola", 20);
        System.out.println("Original Student: " + student);

        String studentJson = JsonParser.toJson(student);
        System.out.println("JSON:\n" + studentJson);

        Student studentDeserialized = JsonParser.fromJson(studentJson,
Student.class);
```

```
        System.out.println("Deserialized Student: " + studentDeserialized);

        System.out.println("\n--- 3. Employee to JSON and back ---");
        Employee employee = new Employee("Andriy", 28);
        System.out.println("Original Employee: " + employee);

        String employeeJson = JsonParser.toJson(employee);
        System.out.println("JSON:\n" + employeeJson);

        Employee employeeDeserialized = JsonParser.fromJson(employeeJson,
Employee.class);
        System.out.println("Deserialized Employee: " + employeeDeserialized);

        System.out.println("\n--- 4. Team<Schoolar> to JSON and back ---");
        Team<Schoolar> schoolarTeam = new Team<>("Dragon");
        schoolarTeam.addNewParticipant(new Schoolar("Ivan", 13));
        schoolarTeam.addNewParticipant(new Schoolar("Mariya", 15));
        System.out.println("Original Team: " + schoolarTeam);

        String schoolarTeamJson = JsonParser.toJson(schoolarTeam);
        System.out.println("JSON:\n" + schoolarTeamJson);

        Team<Schoolar> schoolarTeamDeserialized =
JsonParser.fromJson(schoolarTeamJson, Team.class);
        System.out.println("Deserialized Team: " + schoolarTeamDeserialized);

        System.out.println("\n--- 5. Team<Student> to JSON and back ---");
        Team<Student> studentTeam = new Team<>("Vpered");
        studentTeam.addNewParticipant(new Student("Mykola", 20));
        studentTeam.addNewParticipant(new Student("Viktoria", 21));
        System.out.println("Original Team: " + studentTeam);

        String studentTeamJson = JsonParser.toJson(studentTeam);
        System.out.println("JSON:\n" + studentTeamJson);

        Team<Student> studentTeamDeserialized =
JsonParser.fromJson(studentTeamJson, Team.class);
        System.out.println("Deserialized Team: " + studentTeamDeserialized);

        System.out.println("\n--- 6. Team<Employee> to JSON and back ---");
        Team<Employee> employeeTeam = new Team<>("Robotyagi");
        employeeTeam.addNewParticipant(new Employee("Andriy", 28));
        employeeTeam.addNewParticipant(new Employee("Oksana", 25));
        System.out.println("Original Team: " + employeeTeam);

        String employeeTeamJson = JsonParser.toJson(employeeTeam);
        System.out.println("JSON:\n" + employeeTeamJson);

        Team<Employee> employeeTeamDeserialized =
JsonParser.fromJson(employeeTeamJson, Team.class);
        System.out.println("Deserialized Team: " + employeeTeamDeserialized);

        System.out.println("\n--- 7. Save and Load from JSON file ---");
        Student fileStudent = new Student("John", 22);
        String filePath = "student.json";

        JsonFileOperations.saveToJsonFile(fileStudent, filePath);
        Student loadedStudent = JsonFileOperations.loadFromJsonFile(filePath,
Student.class);
        System.out.println("Loaded Student: " + loadedStudent);

        logger.info("Task_6: JSON Parser completed successfully");
```

```
    }
}
```

Виконання програми:

```
=== ЗАВДАННЯ 2: Серіалізація та Десеріалізація ===

To the team Vpered was added participant Mykola
To the team Vpered was added participant Viktoria

--- Об'єкти ДО серіалізації ---
Schoolar: Schoolar{name='Ivan', age=13, tempInfo='Temporary data for Ivan'}
Student: Student{name='Mykola', age=20, tempInfo='Temporary data for Mykola'}
Employee: Employee{name='Andriy', age=28, tempInfo='Temporary data for Andriy'}
Team: Team{name='Vpered', participants=[Student{name='Mykola', age=20, tempInfo='Temporary data for Mykola'}, Student{name='Viktoria', age=21, tempInfo='Temporary data for Viktoria'}]}
Student tempInfo (до серіалізації): Temporary data for Mykola

--- Серіалізація об'єктів ---
✓ Серіалізовано: schoolar.ser
✓ Серіалізовано: student.ser
✓ Серіалізовано: employee.ser
✓ Серіалізовано: team.ser
Всі об'єкти успішно серіалізовані!

--- Об'єкти ПІСЛЯ десеріалізації ---
✓ Десеріалізовано: schoolar.ser
✓ Десеріалізовано: student.ser
✓ Десеріалізовано: employee.ser
✓ Десеріалізовано: team.ser
Schoolar: Schoolar{name='Ivan', age=13, tempInfo='null'}
Student: Student{name='Mykola', age=20, tempInfo='null'}
Student tempInfo: null (було виключено з серіалізації!)
Employee: Employee{name='Andriy', age=28, tempInfo='null'}
Team: Team{name='Vpered', participants=[Student{name='Mykola', age=20, tempInfo='null'}, Student{name='Viktoria', age=21, tempInfo='null'}]}

--- Аналіз результатів ---
✓ Поля з модифікатором 'transient' НЕ були серіалізовані
✓ Після десеріалізації transient поля мають значення за замовчуванням:
  - tempInfo (String) = null
✓ Всі інші поля були успішно серіалізовані та десеріалізовані
```

Рис. 1 Результат виконання завдання 2

```
To the team Dragon was added participant Ivan
To the team Dragon was added participant Mariya
To the team Vpered was added participant Mykola
To the team Vpered was added participant Viktoria
To the team Alpha was added participant Alex
To the team Alpha was added participant Maria
To the team Beta was added participant John
To the team Beta was added participant Kate
The team Beta is winner!
21:06:52.676 [main] ERROR com.education.ztu.Task_3 - Failed to set age: Age cannot be negative
21:06:52.677 [main] ERROR com.education.ztu.game.Team - Attempt to add null participant to team 'TestTeam'
21:06:52.677 [main] ERROR com.education.ztu.Task_3 - Cannot add null participant: Participant cannot be null
21:06:52.678 [main] ERROR com.education.ztu.Task_3 - This would be FATAL level in Log4j - critical system failure simulation
21:06:52.678 [main] ERROR com.education.ztu.Task_3 - ERROR: Error messages about failures
```

Рис. 2 Результат виконання завдання 3

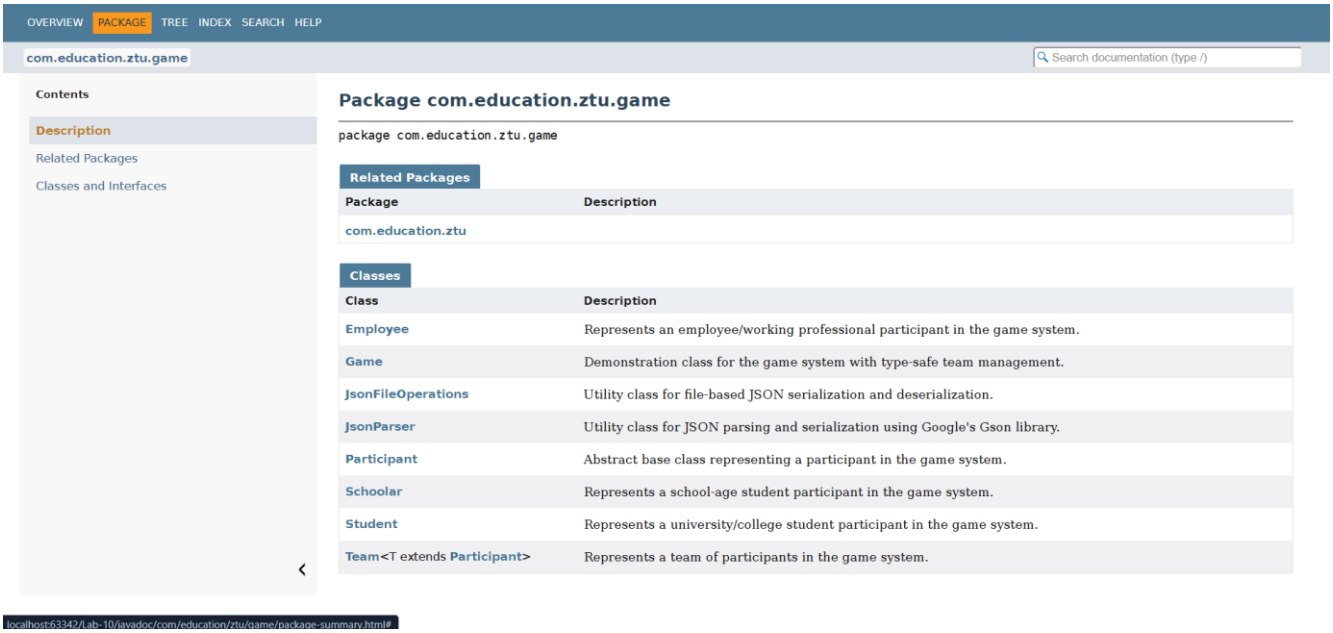

Рис. 3 Результат виконання завдання 4 (створення документації)

```
================================================================================
XML CONTENT
================================================================================
Root element: teams
Total teams: 4

Team #1 [schoolar]
  Name: Dragon
  Rating: 1200
  Wins: 5
  Participants (2):
    - Ivan (Age: 13, Role: Captain)
    - Mariya (Age: 15, Role: Player)

Team #2 [schoolar]
  Name: Rozumnyky
  Rating: 1150
  Wins: 3
  Participants (2):
    - Sergey (Age: 12, Role: Captain)
    - Olga (Age: 14, Role: Player)

Team #3 [student]
  Name: Vpered
  Rating: 1350
  Wins: 8
  Participants (2):
    - Mykola (Age: 20, Role: Captain)
    - Viktoria (Age: 21, Role: Player)

Team #4 [employee]
  Name: Robotyagi
  Rating: 1450
  Wins: 12
  Participants (2):
    - Andriy (Age: 28, Role: Captain)
    - Oksana (Age: 25, Role: Player)
```

Рис. 4 Результат виконання завдання 5

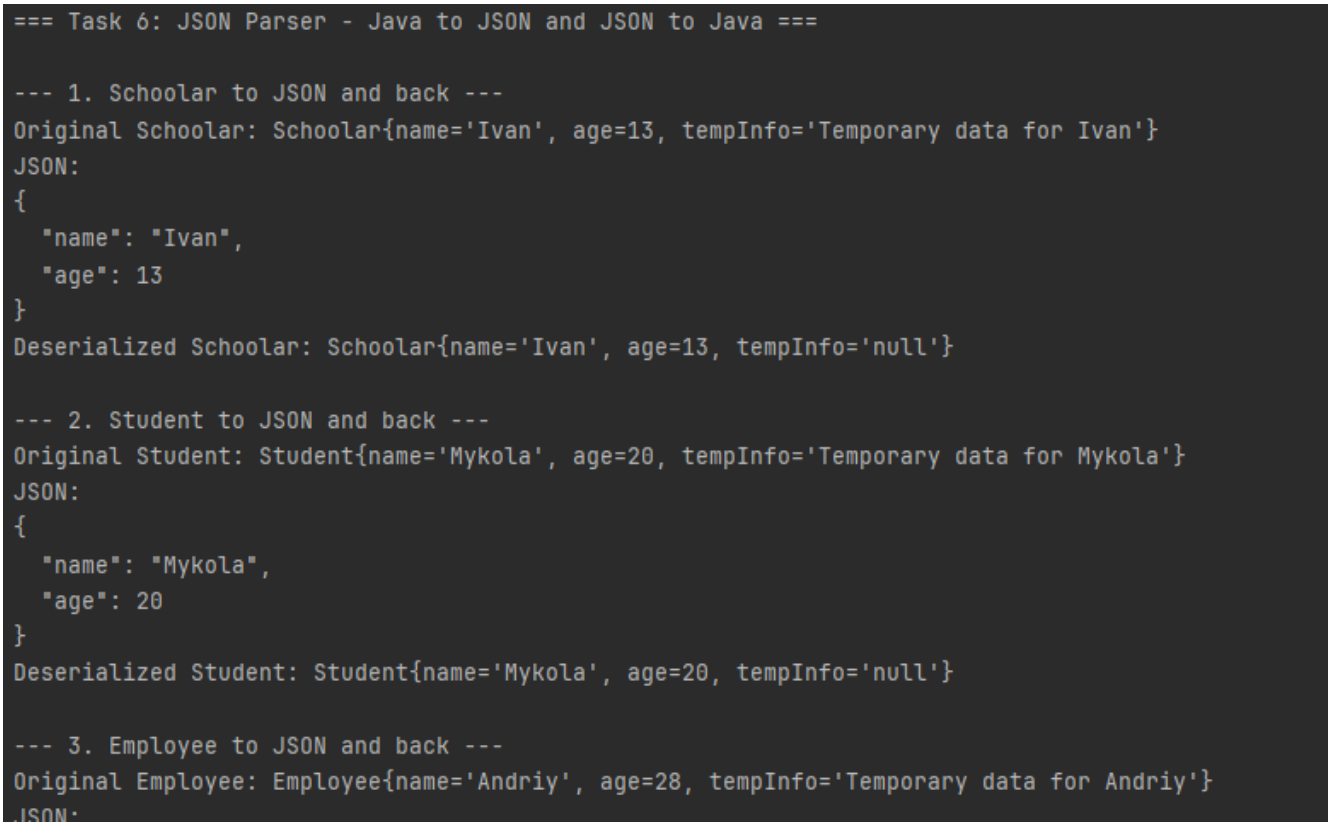

Рис. 5 Результат виконання завдання 6

Посилання на репозиторій: https://github.com/Oleg-Ischuk/Java-IPZ-23-1

Висновок: попрактикувався з XML та JSON парсерами, використанням серіалізації, логуванням та докусентуванням коду.