**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»**

**ІКНІ**

Кафедра **ПЗ**



**ЗВІТ**

до лабораторної роботи №3

**на тему:** *“Робота з текстовою інформацією, файлами та серіалізація”*

**з дисципліни** *“Кросплатформне програмування”*

**Лектор:**

доц. каф. ПЗ

Дяконюк Л. М.

**Виконав:**

ст. гр. ПЗ-33

Юшкевич. А.І.

**Прийняв:**

доц. каф. ПЗ

Баштовий А. В.

«\_\_» \_\_\_\_\_\_\_\_\_ 2024р.

∑=\_\_\_\_\_

Львів – 2024

**Тема.** Робота з текстовою інформацією, файлами та серіалізація.

**Мета.** Навчитися працювати з потоками введення-виведення (I/O Streams), текстовою інформацією, файлами, нативною серіалізацією та сторонніми бібліотеками для серіалізації об'єктів у формати JSON та YAML. Ознайомитися з використанням Maven для управління залежностями та збіркою проекту, вивчити особливості використання систем збірки проєктів.

# Теоретичні відомості

**1.** I/O Streams (Вводу-Виводу потоки):

* Потоки вводу-виводу (I/O Streams) в Java використовуються для читання та запису даних з різних джерел, таких як файли, мережа або системні ресурси. Існують буферизовані потоки, які зменшують кількість фізичних операцій читання/запису, та небуферизовані, які працюють напряму з джерелом даних.
* [Документація Java I/O Streams](https://docs.oracle.com/javase/tutorial/essential/io/streams.html)

**2.** Maven:

* Maven – це популярна система автоматизованої збірки проектів для Java, яка допомагає керувати залежностями, збирати проекти та організовувати їх структуру. Вона використовує файл pom.xml, у якому зазначаються залежності та налаштування проекту.
* [Офіційна документація Maven](https://maven.apache.org/guides/index.html)

**3.** JSON (JavaScript Object Notation):

* JSON – це легкий формат обміну даними, який зручно використовувати для передачі структурованих даних між сервером і клієнтом. Він широко застосовується в веб-розробці для серіалізації об'єктів.
* Документація JSON

**4.** YAML (YAML Ain't Markup Language):

* YAML – це простий у використанні формат серіалізації даних, що підтримує читання та редагування людиною. Він часто використовується для конфігураційних файлів завдяки своїй читабельності та гнучкості.
* [Документація YAML](https://yaml.org/)

**5.** CSV (Comma-Separated Values):

* CSV – це простий формат для зберігання табличних даних, де кожен рядок є записом, а стовпці відокремлені комами. CSV широко використовується для імпорту та експорту даних у базах даних і таблицях.
* Документація CSV

**6.** Maven

* **Maven** – система автоматизованої збірки, яка допомагає керувати проектом та його залежностями через XML конфігурації. **Gradle** – більш сучасна система збірки, яка використовує DSL (Domain Specific Language) для налаштування, дозволяючи більш гнучку конфігурацію проектів.
* [Документація Maven](https://maven.apache.org/)

# Завдання

### **Збірка проектів за допомогою Maven:**

* + Створіть Maven проект.
  + Додайте залежності у файл pom.xml для бібліотек **Gson**, **Jackson**, та **SnakeYAML або альтернативні які потрібні для завдань нижче**.
  + Налаштуйте проект так, щоб використовувати ці бібліотеки для серіалізації/десеріалізації.
  + Виконайте збірку проекту за допомогою команди mvn clean install і переконайтеся, що проект компілюється та працює без помилок.

### **Робота з Java I/O Streams:**

* + Використайте:
    - Небуферизовані потоки (FileInputStream, FileOutputStream) для читання та запису байтів.
    - Буферизовані потоки (BufferedInputStream, BufferedOutputStream) для оптимізації процесу читання та запису.
    - Використайте ObjectInputStream

### **Cеріалізація та десеріалізація об'єктів Java:**

* + Створити клас з кількома полями одне з яких інший клас, а не примітивний тип.
  + Напишіть програму, яка:
    - Створює об'єкт цього класу, серіалізує його в файл
    - Читає з цього файлу та десеріалізує об'єкт
  + Вивести серіалізовані дані та результат десеріалізації

1. **Серіалізація JSON :**
   * Використовуючи бібліотеку **Gson** або **Jackson** або альтернативну, одну з них на вибір:
     + Серіалізуйте об'єкт цього класу у формат JSON та запишіть його у файл.
     + Прочитайте JSON файл, десеріалізуйте його назад в об'єкт Java та виведіть на екран.
   * Переконайтеся, що всі поля класу правильно серіалізовані та десеріалізовані.
2. **Серіалізація YAML:**

Використовуйте бібліотеку **SnakeYAML або альтернативну** для:

* + - Серіалізації об'єкта Java у YAML файл.
    - Читання YAML файлу та десеріалізації його в об'єкт Java.
  + Виведіть об'єкт, що був десеріалізований, для перевірки його коректності.

**Варіант 1:** Управління користувачами соціальної мережі

1. **Робота з потоками вводу-виводу (I/O Streams):**
   * Реалізуйте читання та запис профілів користувачів у файл (поля: ID, ім'я, прізвище, email, список повідомлень).
   * Список повідомлень містить тему, текст і дату.
   * Використовуйте **буферизовані потоки** для збереження профілів, а **небуферизовані** — для збереження повідомлень.
   * **Умова:** Поле email у класі користувача **не повинно бути серіалізоване**.
2. **Нативна серіалізація Java:**
   * Серіалізуйте профілі користувачів у файл, використовуючи стандартну серіалізацію Java.
   * **Умова:** При серіалізації список повідомлень **має залишитися порожнім** і не серіалізуватися.
3. **Серіалізація JSON (GSON або Jackson):**
   * Використайте бібліотеку GSON або Jackson для серіалізації профілів користувачів у JSON.
   * Читання JSON повинно відновлювати всі поля, **крім email**.
4. **Серіалізація YAML (SnakeYAML):**
   * Серіалізуйте профілі користувачів у YAML файл.
   * **Умова:** При десеріалізації відновіть список повідомлень, але **пропустіть дату**.

# Хід роботи

pom.xml:

<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>MavenProject</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <packaging>jar</packaging>  
  
 <name>MavenProject</name>  
 <url>http://maven.apache.org</url>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <maven.compiler.source>23</maven.compiler.source>  
 <maven.compiler.target>23</maven.compiler.target>  
 </properties>  
  
 <dependencies>  
  
 <dependency>  
 <groupId>org.junit.jupiter</groupId>  
 <artifactId>junit-jupiter</artifactId>  
 <version>5.11.3</version>  
 <scope>test</scope>  
 </dependency>  
  
 <dependency>  
 <groupId>com.google.code.gson</groupId>  
 <artifactId>gson</artifactId>  
 <version>2.11.0</version>  
 <scope>provided</scope>  
 </dependency>  
  
 <dependency>  
 <groupId>com.fasterxml.jackson.core</groupId>  
 <artifactId>jackson-databind</artifactId>  
 <version>2.17.2</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.yaml</groupId>  
 <artifactId>snakeyaml</artifactId>  
 <version>1.28</version>  
 </dependency>  
  
 </dependencies>  
  
</project>

Message.java:

package Entities;  
  
import com.google.gson.annotations.Expose;  
import java.io.Serializable;  
import java.time.LocalDateTime;  
import java.time.format.DateTimeFormatter;  
import java.util.Objects;  
  
public class Message implements Serializable {  
 private static final long *serialVersionUID* = 1L;  
 @Expose  
 private String topic;  
 @Expose  
 private String message;  
 @Expose  
 private LocalDateTime date;  
  
 public Message(String topic, String message, LocalDateTime date) {  
 this.topic = topic;  
 this.message = message;  
 this.date = date;  
 }  
  
 public Message(){}  
  
 public String getTopic() {  
 return topic;  
 }  
 public void setTopic(String topic) {  
 this.topic = topic;  
 }  
  
 public String getMessage() {  
 return message;  
 }  
 public void setMessage(String message) {  
 this.message = message;  
 }  
  
 public LocalDateTime getDate() {  
 return date;  
 }  
 public void setDate(LocalDateTime date) {  
 this.date = date;  
 }  
  
 @Override  
 public String toString() {  
 DateTimeFormatter formatter = DateTimeFormatter.*ofPattern*("yyyy-MM-dd HH:mm:ss");  
 return topic + ":\n\n" + message + "\n\ndate: " + date.format(formatter);  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Message message1 = (Message) o;  
 return Objects.*equals*(topic, message1.topic) &&  
 Objects.*equals*(message, message1.message) &&  
 Objects.*equals*(date, message1.date);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(topic, message, date);  
 }  
}

UserProfile.java:

package Entities;  
  
import com.google.gson.annotations.Expose;  
import java.io.Serializable;  
import java.util.List;  
import java.util.Objects;  
import java.util.UUID;  
public class UserProfile implements Serializable {  
 private static final long *serialVersionUID* = 1L;  
 @Expose  
 private String ID = UUID.*randomUUID*().toString();  
 @Expose  
 private String name;  
 @Expose  
 private String surname;  
 @Expose  
 private String email;  
 @Expose  
 private List<Message> messages;  
  
 public UserProfile(String name, String surname, String email, List<Message> messages) {  
 this.name = name;  
 this.surname = surname;  
 this.email = email;  
 this.messages = messages;  
 }  
  
 public UserProfile() {}  
  
 public String getID() {  
 return ID;  
 }  
 public void setID(String ID) {  
 this.ID = ID;  
 }  
  
 public String getName() {  
 return name;  
 }  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public String getSurname() {  
 return surname;  
 }  
 public void setSurname(String surname) {  
 this.surname = surname;  
 }  
  
 public String getEmail() {  
 return email;  
 }  
 public void setEmail(String email) {  
 this.email = email;  
 }  
  
 public List<Message> getMessages() {  
 return messages;  
 }  
 public void setMessages(List<Message> messages) {  
 this.messages = messages;  
 }  
  
 @Override  
 public String toString() {  
 return "Entities.UserProfile{" +  
 "ID=" + ID +  
 ", name='" + name + '\'' +  
 ", surname='" + surname + '\'' +  
 ", email='" + (email != null ? email : "N/A") + '\'' +  
 ", messages=" + messages +  
 '}';  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 UserProfile that = (UserProfile) o;  
 return Objects.*equals*(ID, that.ID);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(ID);  
 }  
}

IOStreams.java:

import Entities.\*;  
  
import java.io.\*;  
import java.util.List;  
  
public class IOStreams {  
 public static void writeMessagesToFile(String filename, List<Message> messages) throws IOException {  
 FileOutputStream fos = new FileOutputStream(filename);  
 ObjectOutputStream oos = new ObjectOutputStream(fos);  
 try {  
 oos.writeObject(messages);  
 }finally {  
 if(fos != null){  
 fos.close();  
 }  
 if(oos != null){  
 oos.close();  
 }  
 }  
 }  
  
 public static List<Message> readMessagesFromFile(String filename) throws IOException, ClassNotFoundException {  
 FileInputStream fis = new FileInputStream(filename);  
 ObjectInputStream ois = new ObjectInputStream(fis);  
 try{  
 return (List<Message>) ois.readObject();  
 }finally {  
 if(fis != null){  
 fis.close();  
 }  
 if(ois != null){  
 ois.close();  
 }  
 }  
 }  
  
 public static void writeUserProfileToFile(String filename, UserProfile userProfile) throws IOException {  
 BufferedOutputStream bos = new BufferedOutputStream(new FileOutputStream(filename));  
 ObjectOutputStream oos = new ObjectOutputStream(bos);  
 try {  
 oos.writeObject(userProfile);  
 }finally {  
 if(bos != null){  
 bos.close();  
 }  
 if(oos != null){  
 oos.close();  
 }  
 }  
 }  
  
 public static UserProfile readUserProfileFromFile(String filename) throws IOException, ClassNotFoundException {  
 BufferedInputStream bis = new BufferedInputStream(new FileInputStream(filename));  
 ObjectInputStream ois = new ObjectInputStream(bis);  
 try {  
 return (UserProfile) ois.readObject();  
 }finally {  
 if(bis != null){  
 bis.close();  
 }  
 if(ois != null){  
 ois.close();  
 }  
 }  
 }  
}

GsonSerializer.java:

import Entities.UserProfile;  
import com.google.gson.Gson;  
import com.google.gson.GsonBuilder;  
import java.io.FileReader;  
import java.io.FileWriter;  
import java.io.IOException;  
import java.io.Reader;  
import java.io.Writer;  
  
public class GsonSerializer {  
  
 private static final Gson *gson* = new GsonBuilder()  
 .setPrettyPrinting()  
 .registerTypeAdapter(UserProfile.class, new UserProfileAdapter())  
 .create();  
  
 public static <T> void serialize(T object, String filePath) throws IOException {  
 Writer writer = new FileWriter(filePath);  
 try{  
 *gson*.toJson(object, writer);  
 }finally {  
 writer.close();  
 }  
 }  
  
 public static <T> T deserialize(String filePath, Class<T> clazz) throws IOException {  
 Reader reader = new FileReader(filePath);  
 try{  
 return *gson*.fromJson(reader, clazz);  
 }finally {  
 reader.close();  
 }  
 }  
}

UserProfileAdapter.java:

import Entities.\*;  
import com.google.gson.TypeAdapter;  
import com.google.gson.stream.JsonReader;  
import com.google.gson.stream.JsonWriter;  
  
import java.io.IOException;  
import java.time.LocalDateTime;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.UUID;  
  
public class UserProfileAdapter extends TypeAdapter<UserProfile> {  
  
 @Override  
 public void write(JsonWriter out, UserProfile userProfile) throws IOException {  
 out.beginObject();  
  
 out.name("ID").value(userProfile.getID());  
 out.name("name").value(userProfile.getName());  
 out.name("surname").value(userProfile.getSurname());  
  
 // Explicitly serialize transient fields  
 if (userProfile.getEmail() != null) {  
 out.name("email").value(userProfile.getEmail());  
 }else {  
 out.name("email").nullValue();  
 }  
 if (userProfile.getMessages() != null) {  
 out.name("messagesCount").value(userProfile.getMessages().size());  
 out.name("messages").beginArray();  
 for (Message message : userProfile.getMessages()) {  
 out.beginObject();  
 out.name("topic").value(message.getTopic());  
 out.name("message").value(message.getMessage());  
 out.name("date").value(message.getDate().toString());  
 out.endObject();  
 }  
 out.endArray();  
 } else {  
 out.name("messages").nullValue();  
 }  
  
 out.endObject();  
 }  
  
 @Override  
 public UserProfile read(JsonReader in) throws IOException {  
 // Basic deserialization logic (skipping the `email` field)  
 in.beginObject();  
  
 String name = null;  
 String surname = null;  
 String ID = null;  
 List<Message> messages = null;  
 int messagesCount = 0;  
  
 while (in.hasNext()) {  
 String fieldName = in.nextName();  
 switch (fieldName) {  
 case "ID":  
 ID = in.nextString();  
 break;  
 case "name":  
 name = in.nextString();  
 break;  
 case "surname":  
 surname = in.nextString();  
 break;  
 case "messagesCount":  
 messagesCount = in.nextInt();  
 break;  
 case "messages":  
 messages = readMessages(in, messagesCount); // Helper method to read messages array  
 break;  
 default:  
 in.skipValue(); // Skip any fields not handled  
 break;  
 }  
 }  
  
 in.endObject();  
 return new UserProfile(name, surname, null, messages); // `email` is not deserialized  
 }  
  
 private List<Message> readMessages(JsonReader in, int messagesCount) throws IOException {  
 in.beginArray();  
 List<Message> messages = new ArrayList<>();  
  
 for (int i = 0; i < messagesCount; i++) {  
 in.beginObject();  
 String topic = null;  
 String message = null;  
 LocalDateTime date = null;  
  
 while(topic == null || message == null || date == null) {  
 switch (in.nextName()) {  
 case "topic":  
 topic = in.nextString();  
 break;  
 case "message":  
 message = in.nextString();  
 break;  
 case "date":  
 date = LocalDateTime.*parse*(in.nextString());  
 break;  
 default:  
 in.skipValue();  
 }  
 }  
 in.endObject();  
  
 messages.add(new Message(topic, message, date));  
 }  
 in.endArray();  
  
 return messages;  
 }  
}

YamlSerializer.java:

import Entities.UserProfile;  
import org.yaml.snakeyaml.DumperOptions;  
import org.yaml.snakeyaml.TypeDescription;  
import org.yaml.snakeyaml.Yaml;  
import org.yaml.snakeyaml.nodes.Tag;  
import org.yaml.snakeyaml.representer.Representer;  
  
import java.io.FileInputStream;  
import java.io.FileWriter;  
import java.io.IOException;  
import java.io.InputStream;  
import java.util.UUID;  
  
public class YamlSerializer {  
  
 private static final Yaml *yaml*;  
  
 static {  
 var options = new DumperOptions();  
 options.setIndent(4);  
 options.setPrettyFlow(true);  
 options.setDefaultFlowStyle(DumperOptions.FlowStyle.*BLOCK*);  
  
 var representer = new Representer(options);  
 representer.getPropertyUtils().setSkipMissingProperties(true); // Пропуск відсутніх властивостей  
 representer.addClassTag(UserProfile.class, Tag.*MAP*);  
 representer.addClassTag(UUID.class, Tag.*STR*);  
  
 *yaml* = new Yaml(representer);  
 }  
  
 public static <T> void serialize(T object, String filePath) throws IOException {  
 try (FileWriter writer = new FileWriter(filePath)) {  
 *yaml*.dump(object, writer);  
 }  
 }  
  
 public static <T> T deserialize(String filePath, Class<T> clazz) throws IOException {  
 try (InputStream input = new FileInputStream(filePath)) {  
 return *yaml*.loadAs(input, clazz);  
 }  
 }  
}

IOStreamsTest.java:

import Entities.Message;  
import Entities.UserProfile;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
import java.io.IOException;  
import java.time.LocalDateTime;  
import java.time.Month;  
import java.util.ArrayList;  
import java.util.\*;  
import static org.junit.jupiter.api.Assertions.\*;  
  
public class IOStreamsTest {  
  
 private static final String *messagesFile* = "MessagesFile.bin";  
 private static final String *userProfileFile* = "UserProfileFile.bin";  
 private List<Message> messages;  
 private UserProfile userProfile;  
  
 @BeforeEach  
 void setUp() {  
 messages = List.*of*(  
 new Message("Review", "Lorem Ipsum", LocalDateTime.*of*(2022, Month.*JANUARY*, 4, 0, 0)),  
 new Message("Buzzkills in New York", "I hate buzzkills. They're funny tho.", LocalDateTime.*of*(2022, Month.*JANUARY*, 4, 0, 0)),  
 new Message("Summary", "I want to finally sum up all the summaries and results and make last conclusion", LocalDateTime.*of*(2022, Month.*JANUARY*, 4, 0, 0))  
 );  
  
 userProfile = new UserProfile("Andrii", "Kmitliviy", "akmitliviy@gmail.com", messages);  
 }  
  
 @Test  
 void testWriteMessagesToFile() {  
  
 try {  
 IOStreams.*writeMessagesToFile*(*messagesFile*, messages);  
 *assertTrue*(true);  
 } catch (IOException e) {  
 System.*out*.println(e);  
 *fail*();  
 }  
 }  
  
 @Test  
 void testReadMessagesFromFile() {  
 List<Message> readMessages = new ArrayList<>();  
 try{  
 readMessages = IOStreams.*readMessagesFromFile*(*messagesFile*);  
 }catch (IOException | ClassNotFoundException e){  
 System.*out*.println(e);  
 *fail*();  
 }  
  
 *assertEquals*(readMessages.size(), messages.size());  
 for (Message message : readMessages) {  
 System.*out*.println(message + "\n\*\*\*\n\n");  
 }  
 }  
  
 @Test  
 void testWriteUserProfileToFile() {  
 try{  
 IOStreams.*writeUserProfileToFile*(*userProfileFile*, userProfile);  
 *assertTrue*(true);  
 }catch (IOException e){  
 System.*out*.println(e);  
 *fail*();  
 }  
 }  
  
 @Test  
 void testReadUserProfileFromFile() {  
 UserProfile readUserProfile = null;  
 try{  
 readUserProfile = IOStreams.*readUserProfileFromFile*(*userProfileFile*);  
 }catch (IOException | ClassNotFoundException e){  
 System.*out*.println(e);  
 *fail*();  
 }  
  
 *assertEquals*(readUserProfile.getName(), userProfile.getName());  
 *assertEquals*(readUserProfile.getSurname(), userProfile.getSurname());  
 *assertNotEquals*(readUserProfile.getEmail(), userProfile.getEmail());  
 //assertEquals(readUserProfile.getMessages().toString(), messages.toString());  
  
 System.*out*.println(readUserProfile);  
  
 }  
}

GsonSerializerTest.java:

import Entities.\*;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
  
import java.io.IOException;  
import java.time.LocalDateTime;  
import java.time.Month;  
import java.util.ArrayList;  
import java.util.List;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
class GsonSerializerTest {  
  
 private UserProfile userProfile;  
 private static final String *FILE\_PATH* = "GsonFile.json";  
  
 @BeforeEach  
 void setUp() {  
 List<Message> messages = new ArrayList<>();  
 messages.add(new Message("Hello", "This is the first message", LocalDateTime.*now*()));  
 messages.add(new Message("Buzzkills in New York", "I hate buzzkills. They're funny tho.", LocalDateTime.*of*(2022, Month.*JANUARY*, 4, 0, 0)));  
 userProfile = new UserProfile("John", "Doe", "john.doe@example.com", messages);  
 }  
  
 @Test  
 void testSerialization() throws IOException {  
 // Serialize user profile to JSON  
 GsonSerializer.*serialize*(userProfile, *FILE\_PATH*);  
  
 // Read the JSON file content as a string for validation  
 String jsonContent = new String(java.nio.file.Files.*readAllBytes*(java.nio.file.Paths.*get*(*FILE\_PATH*)));  
  
 // Check that email is present in JSON (since we customized serialization to include transient)  
 *assertTrue*(jsonContent.contains("john.doe@example.com"), "Email should be serialized even if transient");  
 }  
  
 @Test  
 void testDeserialization() throws IOException {  
 // Serialize and then Deserialize user profile  
 GsonSerializer.*serialize*(userProfile, *FILE\_PATH*);  
 UserProfile deserializedUserProfile = GsonSerializer.*deserialize*(*FILE\_PATH*, UserProfile.class);  
  
 // Check that deserialized object's email is null (excluded from deserialization)  
 *assertNull*(deserializedUserProfile.getEmail(), "Email should not be deserialized");  
  
 // Check other fields  
 *assertEquals*(userProfile.getName(), deserializedUserProfile.getName());  
 *assertEquals*(userProfile.getSurname(), deserializedUserProfile.getSurname());  
  
 *assertEquals*(userProfile.getMessages(), deserializedUserProfile.getMessages());  
 }  
}

YamlSerializerTest.java:

import Entities.\*;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
  
import java.io.IOException;  
import java.time.LocalDateTime;  
import java.util.ArrayList;  
import java.util.List;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
class YamlSerializerTest {  
  
 private UserProfile userProfile;  
 private static final String *FILE\_PATH* = "YamlFile.yaml";  
  
 @BeforeEach  
 void setUp() {  
 List<Message> messages = new ArrayList<>();  
 messages.add(new Message("Hello", "This is the first message", LocalDateTime.*now*()));  
 userProfile = new UserProfile("John", "Doe", "john.doe@example.com", messages);  
 }  
  
 @Test  
 void testSerialization() throws IOException {  
 // Serialize UserProfile to YAML  
 YamlSerializer.*serialize*(userProfile, *FILE\_PATH*);  
  
 // Check that file is created and has content  
 String yamlContent = new String(java.nio.file.Files.*readAllBytes*(java.nio.file.Paths.*get*(*FILE\_PATH*)));  
 *assertTrue*(yamlContent.contains("John"), "Name should be serialized");  
 *assertTrue*(yamlContent.contains("Doe"), "Surname should be serialized");  
 *assertTrue*(yamlContent.contains("Hello"), "Messages should be serialized");  
 }  
  
 @Test  
 void testDeserialization() throws IOException {  
 // Serialize and then Deserialize UserProfile  
 YamlSerializer.*serialize*(userProfile, *FILE\_PATH*);  
 UserProfile deserializedUserProfile = YamlSerializer.*deserialize*(*FILE\_PATH*, UserProfile.class);  
  
 // Check deserialized data  
 *assertEquals*(userProfile.getName(), deserializedUserProfile.getName());  
 *assertEquals*(userProfile.getSurname(), deserializedUserProfile.getSurname());  
  
 // Check that date is skipped in messages after deserialization  
 *assertNotNull*(deserializedUserProfile.getMessages());  
 *assertEquals*(1, deserializedUserProfile.getMessages().size());  
 *assertNull*(deserializedUserProfile.getMessages().get(0).getDate(), "Date should be skipped in deserialization");  
 }  
}

Додаю [посилання](https://github.com/Akmitliviy/CPP) на GitHub репозиторій.

**Висновки:** протягом виконання цієї лабораторної роботи я навчився використовувати основний інструментарій мови для пошуку, зміни, форматування текстової інформації.