МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАНИЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» ФАКУЛЬТЕТ АВТОМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Кафедра вычислительной техники

Лабораторная работа №2 по дисциплине «Инжиринг программного обеспечения»

**Рефлексия и аннотации.**

Группа: АВТ-342

Студент: Вострецов Н.С., Фадеев В.А.

Вариант: 5

Преподаватель: Дыминский И. И.

Новосибирск 2025

Содержание

[Постановка задачи 2](#_Toc210593035)

[Цель работы 2](#_Toc210593036)

[Ход работы 2](#_Toc210593037)

[Теоретические основы 2](#_Toc210593038)

[Рефлексия 2](#_Toc210593039)

[Аннотации 3](#_Toc210593040)

[Сериализация 4](#_Toc210593041)

[Десериализация 4](#_Toc210593042)

[Структура проекта 5](#_Toc210593043)

[Реализация 6](#_Toc210593044)

[@XmlField 6](#_Toc210593045)

[Модели данных 6](#_Toc210593046)

[Класс XmlSerializerService 7](#_Toc210593047)

[serializeObject 8](#_Toc210593048)

[Класс ConsoleView 9](#_Toc210593049)

[Контроллер AppController 9](#_Toc210593050)

[Тестовые примеры выполнения программы 9](#_Toc210593051)

[Тестирование 12](#_Toc210593052)

[Вывод тестирования 15](#_Toc210593053)

[Вывод 16](#_Toc210593054)

[Листинг демонстрационной программы 17](#_Toc210593055)

# Постановка задачи

С использование средств рефлексии разработать универсальный сервис, применимый для объектов произвольного класса. Передается имя класса или его описатель класса Class. Разработать универсальный сервис, использующий аннотированные элементы класса.

**Вариант 5**

Собственный XML-сериализатор с сериализацией ссылок и массивов ссылок.

# Цель работы

Целью выполнения лабораторной работы является освоение механизмов аннотаций и рефлексии в языке Java, а также их практическое применение при создании универсального XML-сериализатора и XML-десериализатора, способных работать с объектами произвольных классов, в том числе содержащих массивы и вложенные ссылки на другие объекты.

# Ход работы

Проект реализован с использованием Java 23 и системы сборки Maven. Его структура включает аннотацию XmlField, сервис XmlSerializerService, контроллер AppController, консольное представление ConsoleView, а также модельные классы Person, Team и Company, которые служат для проверки корректности работы сервиса. В классе Person описаны поля имени и возраста, в классе Team хранится название команды и массив участников, а в классе Company содержится название компании и ссылка на объект Team.

# Теоретические основы

## Рефлексия

**Рефлексия** — это механизм в Java, который позволяет **программе анализировать собственную структуру и изменять поведение объектов во время выполнения.**Иными словами, с помощью рефлексии можно узнать, **какие поля, методы, аннотации и конструкторы есть у класса,** получить к ним доступ, даже если они **private**, а также вызывать методы или изменять значения полей без их прямого упоминания в коде.

С помощью неё программа анализирует произвольный объект, находит его поля, проверяет, какие из них помечены аннотацией @XmlField, и затем динамическиизвлекаетихзначения, не зная заранее, из какого класса этот объект.

for (Field field : obj.getClass().getDeclaredFields()) {

if (field.isAnnotationPresent(XmlField.class)) {

field.setAccessible(true);

Object value = field.get(obj); // получить значение поля

// далее сериализация значения в XML

}

}  
  
Это позволяет создать новый класс и XmlSerializerService сможет **сразу** сериализовать и десериализовать его, без необходимости писать отдельный код.

## Аннотации

Аннотации в Java представляют собой **метаданные**, то есть дополнительные сведения, которые можно прикреплять к элементам программы — классам, методам, полям, конструкторам, параметрам и даже самим аннотациям.

В данной лабораторной работе была создана **собственная аннотация** @XmlField, предназначенная для указания того, какие поля класса необходимо сериализовать и десериализовать. Такая аннотация позволяет разработчику отметить только нужные поля, избавляя программу от необходимости обрабатывать всё подряд. Это делает сериализацию более гибкой, настраиваемой и универсальной.

@Retention(RetentionPolicy.RUNTIME)

@Target(ElementType.FIELD)

public @interface XmlField {

String name() default "";

}

Элемент @Retention(RetentionPolicy.RUNTIME) сообщает компилятору и виртуальной машине Java, что аннотация должна сохраняться не только в исходном коде и байт-коде, но и быть доступной во время выполнения программы. Это необходимо, чтобы можно было получить доступ к аннотации через Reflection API. Без этой директивы аннотация не была бы видна в рантайме, и программа не смогла бы определить, какие поля нужно сериализовать. Элемент @Target(ElementType.FIELD) указывает, что данная аннотация может применяться только к полям класса.

## Сериализация

Сериализация — это процесс преобразования объекта программы в последовательность байтов или строковое представление, пригодное для хранения или передачи. Основная идея заключается в том, чтобы **сохранить текущее состояние объекта** — значения его полей и вложенных объектов — в таком виде, который можно затем **восстановить обратно**, создав идентичный объект в памяти.

**XML-сериализация** представляет собой преобразование объекта в иерархическую структуру тегов, где каждый тег соответствует полю объекта, а его значение заключено между открывающим и закрывающим тегом.

## Десериализация

Десериализация — это процесс, обратный сериализации, заключающийся в **восстановлении объекта программы из его сохранённого представления**. Если сериализация «упаковывает» объект в строку или поток байтов, то десериализация, наоборот, «распаковывает» эти данные и создаёт в памяти новую копию исходного объекта с теми же значениями полей.

# Структура проекта

**Изображение выглядит как текст, снимок экрана, диаграмма, линия

Содержимое, созданное искусственным интеллектом, может быть неверным.**

Рис. 1. — UML диаграмма.

**Изображение выглядит как текст, снимок экрана, программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.**

Рис. 2. — Структура проекта.

Сборка и запуск:

make compile, make run, make test, make clean

# Реализация

## @XmlField

Для управления процессом сериализации и десериализации в программе используется собственная аннотация @XmlField.  
Она служит для того, чтобы указать, какие именно поля класса должны сохраняться в XML-файл и считываться обратно при восстановлении объекта.

Параметр name задаёт название XML-тега, под которым будет записано значение поля.  
Если его не указать, сериализатор использует имя самого поля.

Пример использования аннотации в классе:

public class Person {

@XmlField(name = "name")

private String name;

@XmlField(name = "age")

private int age;

}

## Модели данных

Person.java

public class Person {

@XmlField(name = "name")

private String name;

@XmlField(name = "age")

private int age;

}

Team.java

public class Team {

@XmlField(name = "teamName")

private String teamName;

@XmlField(name = "members")

private Person[] members;

}

Company.java

public class Company {

@XmlField(name = "companyName")

private String companyName;

@XmlField(name = "mainTeam")

private Team mainTeam;

}

## Класс XmlSerializerService

XmlSerializerService — это класс, он умеет превратить любой помеченный аннотациями объект в XML (сериализация) и восстановить объект из XML (десериализация). Основа — **рефлексия**: сервис находит поля с @XmlField, читает/пишет их значения и при необходимости рекурсивно обходит вложенные объекты и массивы.

Как идёт сериализация (в файл):

Создаётся пустой DOM-документ, корневой элемент — имя класса объекта. Для каждого поля с @XmlField берётся имя тега: если в аннотации name пустой — берётся имя поля. По типу значения выбирается ветка:

простые типы/строки → текст в теге, массив → перебор элементов внутри контейнера, вложенный объект → рекурсивный вызов.

Как идёт десериализация (из файла):

XML читается в DOM-дерево. Создаётся пустой экземпляр нужного класса (через clazz.getDeclaredConstructor().newInstance()). Для каждого поля с @XmlField ищется одноимённый дочерний элемент в XML. По типу поля:

String/int/boolean → читается текст и приводится к типу, массив → перебираются дочерние узлы, каждый десериализуется рекурсивно и собирается в массив, вложенный объект → рекурсивная десериализация.

## serializeObject

Метод serializeObject реализует универсальную сериализацию объекта Java в формат XML. Он рекурсивно обходит все поля объекта, отмеченные специальной аннотацией @XmlField, и формирует соответствующие XML-теги с содержимым.

Алгоритм работы метода можно описать следующим образом:

1. **Проверка входных данных**:  
   Если объект равен null, сериализация не выполняется.
2. **Определение класса объекта**:  
   С помощью рефлексии метод получает список всех полей класса и обрабатывает только те из них, которые помечены аннотацией @XmlField.
3. **Определение имени XML-тега**:  
   Имя тэга берётся из параметра аннотации @XmlField(name="..."). Если оно не указано, используется имя поля в классе.
4. **Создание XML-элемента**:  
   Для каждого поля создаётся XML-тег с найденным именем.
5. **Обработка значения поля**:

Если значение null, в XML добавляется пустой тег.Если значение является простым типом (String, Number, Boolean, Character), оно преобразуется в строку и вставляется внутрь тега.

Если значение является массивом или коллекцией, каждый элемент оборачивается во вложенный тег <item>. Для простых элементов в <item> помещается текст, для объектов запускается рекурсивная сериализация.

Если значение является объектом, метод рекурсивно сериализует его поля и вкладывает их внутрь соответствующего тега.

1. **Формирование дерева XML**:  
   Сформированный элемент добавляется к родительскому узлу. Таким образом строится иерархическое XML-дерево, отражающее структуру объекта и его вложенных элементов.

## Класс ConsoleView

ConsoleView — это простой консольный интерфейс, через который пользователь взаимодействует с приложением. Его задачи: показать меню и получить выбор пользователя (сериализация или десериализация); собрать данные о компании с клавиатуры (имя компании, название команды, участники); передать объект в XmlSerializerService для сохранения в XML и вывести результат на экран; прочитать XML из файла, восстановить объект и показать его в человекочитаемом виде.

## Контроллер AppController

AppController — это центральный управляющий класс приложения, который связывает между собой пользовательский интерфейс (ConsoleView) и логику работы с данными (XmlSerializerService). Именно он координирует процесс сериализации и десериализации, получая объекты от пользователя, передавая их сервису и управляя выводом результатов.

# Тестовые примеры выполнения программы

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рис. 1. — Запуск программы.

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рис. 2. — Ввод значений.

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рис. 3. — XML файл.

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рис. 4. — Загрузка параметров из XML файла.

# Тестирование

Для проверки корректности работы разработанного сервиса сериализации и десериализации был создан модульный тест XmlSerializerServiceTest на базе библиотеки **JUnit 5 (Jupiter).**  
Этот тест автоматически проверяет, что объект с вложенными структурами корректно преобразуется в XML-файл, а затем полностью восстанавливается из него без потери данных.

В тесте моделируется полная рабочая ситуация — сериализация и десериализация трёх взаимосвязанных классов: Company, Team и Person.  
Сначала создаются объекты с реальными значениями, затем сервис записывает их в XML-файл, после чего этот файл читается обратно, и тест сравнивает восстановленные данные с исходными.

**Создание исходных данных.**  
Тест начинает с создания двух объектов Person — *Анна* (28 лет) и *Иван* (32 года).  
Они добавляются в массив участников и передаются в объект Team с названием *Разработка*. Затем создаётся объект Company с названием *ТехноЛаб*, в который помещается команда.

Таким образом формируется трёхуровневая структура:  
Company → Team → Person[]

При помощи вызова

XmlSerializerService.serialize(originalCompany, xmlFile.getAbsolutePath());

Объект Company и все вложенные в него данные сериализуются в XML-файл company.xml внутри временной директории (tempDir).  
В процессе сериализации используется рефлексия: XmlSerializerService анализирует все поля, помеченные аннотацией @XmlField, и записывает их в виде тегов XML.

**Десериализация обратно в объект.**

Далее выполняется обратное преобразование:

Company restoredCompany = XmlSerializerService.deserialize(Company.class, xmlFile.getAbsolutePath());

Сервис считывает XML-файл, создаёт новый объект Company и с помощью Reflection API заполняет его поля.  
Все вложенные объекты (Team и Person[]) восстанавливаются рекурсивно.

После восстановления данные сравниваются с исходными при помощи утверждений (assertEquals):

название компании (ТехноЛаб); название команды (Разработка); количество участников (2); имя и возраст каждого участника.

Если хотя бы одно значение не совпадает, тест падает, что сигнализирует об ошибке в сериализации или десериализации.

Данный тест охватывает сразу несколько аспектов работы программы:

| **Проверяемая особенность** | **Что подтверждает** |
| --- | --- |
| Поддержка вложенных объектов | Сервис корректно сериализует и десериализует объект Company, содержащий Team и массив Person[]. |
| Корректность аннотаций @XmlField | Обрабатываются только поля, помеченные аннотацией. |
| Работа с массивами ссылок | Массив Person[] сохраняется в XML через теги <item> и восстанавливается обратно. |
| Корректное преобразование типов | Строки и числовые значения (age) сохраняются без искажений. |
| Поддержка кириллицы | XML корректно обрабатывает символы русского языка (например, “ТехноЛаб”, “Анна”, “Иван”). |
| Полная симметричность операций | После цикла «сериализация → десериализация» объект полностью идентичен исходному. |

Изображение выглядит как текст, снимок экрана

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рис. 5. — Результаты тестирования.

## Вывод тестирования

Проведённый тест XmlSerializerServiceTest доказал корректность работы сервиса сериализации и десериализации.  
Объекты произвольных классов, содержащие вложенные структуры и массивы ссылок, сохраняются и восстанавливаются без потерь.  
XML-файл создаётся корректно, аннотации @XmlField отрабатывают как задумано, а кириллические данные обрабатываются без ошибок.

Таким образом, реализация полностью соответствует заданию лабораторной работы и подтверждает, что разработанный универсальный XML-сериализатор на основе рефлексии и аннотаций функционирует правильно.

# Вывод

Входе выполнения лабораторной работы была проделана комплексная работа по изучению и практическому применению механизмов рефлексии и аннотаций в языке программирования Java. В процессе выполнения задания особое внимание уделялось пониманию того, каким образом можно использовать встроенные средства языка для анализа структуры произвольных классов и динамического доступа к их полям во время выполнения программы.

В результате была создана полноценная система сериализации и десериализации, реализованная в виде XML-сервисa, который позволяет сохранять и восстанавливать состояние объектов произвольных классов. Реализация базируется на применении аннотаций, определяющих, какие именно поля должны участвовать в процессе сериализации, и на механизмах рефлексии, обеспечивающих динамический доступ к этим полям. Благодаря такому подходу программа может работать с любыми структурами данных, в том числе со сложными иерархическими объектами, не требуя для них написания специализированного кода. Сервис одинаково успешно обрабатывает как простые типы данных, так и массивы, а также вложенные объекты, корректно формируя их XML-представление и обеспечивая симметричное восстановление исходной структуры при десериализации.

Важной частью работы стало тестирование разработанного функционала. Для этого использовалась библиотека **JUnit 5**, которая позволила создать автоматические модульные тесты и убедиться в корректности работы всех ключевых операций. Проведённые испытания подтвердили, что сервис корректно сохраняет и восстанавливает объекты различных типов, включая вложенные структуры и массивы, при этом полностью сохраняются значения всех полей, структура XML остаётся читаемой и соответствующей исходной модели данных. Отдельное внимание уделялось проверке работы с кириллическими символами, что продемонстрировало совместимость решения с локализованными данными.

# Листинг демонстрационной программы

**XmlField.java:**

package org.example.annotations;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)

@Target(ElementType.FIELD)

public @interface XmlField {

String name() default "";

}

**AppController.java**

package org.example.controllers;

import org.example.models.\*;

import org.example.services.XmlSerializerService;

import org.example.views.ConsoleView;

public class AppController {

private final ConsoleView view = new ConsoleView();

public void run() {

boolean running = true;

while (running) {

int choice = view.showMenu();

switch (choice) {

case 1 -> handleInputAndSave();

case 2 -> handleLoad();

case 3 -> {

running = false;

view.showMessage("Выход из программы...");

}

default -> view.showMessage("Неверный выбор, попробуйте снова!");

}

}

}

private void handleInputAndSave() {

String companyName = view.readString("Введите название компании: ");

String teamName = view.readString("Введите название команды: ");

int memberCount = view.readInt("Введите количество участников команды: ");

Person[] members = new Person[memberCount];

for (int i = 0; i < memberCount; i++) {

String name = view.readString("Имя участника #" + (i + 1) + ": ");

int age = view.readInt("Возраст участника #" + (i + 1) + ": ");

members[i] = new Person(name, age);

}

Team team = new Team(teamName, members);

Company company = new Company(companyName, team);

try {

XmlSerializerService.serialize(company, "company.xml");

view.showMessage("Данные успешно сохранены в company.xml");

} catch (Exception e) {

view.showMessage("Ошибка при сохранении: " + e.getMessage());

}

}

private void handleLoad() {

try {

Company company = XmlSerializerService.deserialize(Company.class, "company.xml");

view.showMessage("Данные загружены из company.xml:");

view.showMessage("Компания: " + company.getCompanyName());

view.showMessage("Команда: " + company.getMainTeam().getTeamName());

for (Person p : company.getMainTeam().getMembers()) {

view.showMessage(" - " + p.getName() + ", возраст " +

}

} catch (Exception e) {

view.showMessage("Ошибка при загрузке: " + e.getMessage());

}

}

}

**Company.java**

package org.example.models;

import org.example.annotations.XmlField;

public class Company {

@XmlField(name = "companyName")

private String companyName;

@XmlField(name = "mainTeam")

private Team mainTeam;

public Company() {}

public Company(String companyName, Team mainTeam) {

this.companyName = companyName;

this.mainTeam = mainTeam;

}

public String getCompanyName() { return companyName; }

public Team getMainTeam() { return mainTeam; }

}

**Person.java**

package org.example.models;

import org.example.annotations.XmlField;

public class Person {

@XmlField(name = "name")

private String name;

@XmlField(name = "age")

private int age;

public Person() {}

public Person(String name, int age) {

this.name = name;

this.age = age;

}

public String getName() { return name; }

public int getAge() { return age; }

}

**Team.java**

package org.example.models;

import org.example.annotations.XmlField;

public class Team {

@XmlField(name = "teamName")

private String teamName;

@XmlField(name = "members")

private Person[] members;

public Team() {}

public Team(String teamName, Person[] members) {

this.teamName = teamName;

this.members = members;

}

public String getTeamName() { return teamName; }

public Person[] getMembers() { return members; }

}

**XmlSerializerService.java**

package org.example.services;

import org.example.annotations.XmlField;

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;

import java.lang.reflect.Array;

import java.lang.reflect.Field;

import java.lang.reflect.Array;

import java.util.Collection;

public class XmlSerializerService {

public static void serialize(Object obj, String filename) throws Exception {

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

DocumentBuilder builder = factory.newDocumentBuilder();

Document doc = builder.newDocument();

Element root = doc.createElement(obj.getClass().getSimpleName());

doc.appendChild(root);

serializeObject(obj, root, doc);

TransformerFactory tf = TransformerFactory.newInstance();

Transformer transformer = tf.newTransformer();

transformer.setOutputProperty(OutputKeys.INDENT, "yes");

transformer.setOutputProperty("{http://xml.apache.org/xslt}indent-amount", "2");

transformer.transform(new DOMSource(doc), new StreamResult(new File(filename)));

}

private static void serializeObject(Object obj, Element parent, Document doc) throws Exception {

if (obj == null) return;

Class<?> clazz = obj.getClass();

for (Field field : clazz.getDeclaredFields()) {

field.setAccessible(true);

if (!field.isAnnotationPresent(XmlField.class)) continue;

Object value = field.get(obj);

String tagName = field.getAnnotation(XmlField.class).name();

if (tagName.isEmpty()) tagName = field.getName();

Element elem = doc.createElement(tagName);

if (value == null) {

parent.appendChild(elem);

continue;

}

Class<?> vType = value.getClass();

if (vType.isArray()) {

int len = Array.getLength(value);

for (int i = 0; i < len; i++) {

Object item = Array.get(value, i);

Element itemElem = doc.createElement("item");

if (item == null) {

elem.appendChild(itemElem);

continue;

}

if (isScalar(item)) {

itemElem.setTextContent(item.toString());

} else {

serializeObject(item, itemElem, doc);

}

elem.appendChild(itemElem);

}

} else if (value instanceof Collection<?> col) {

for (Object item : col) {

Element itemElem = doc.createElement("item");

if (item == null) {

elem.appendChild(itemElem);

continue;

}

if (isScalar(item)) {

itemElem.setTextContent(item.toString());

} else {

serializeObject(item, itemElem, doc);

}

elem.appendChild(itemElem);

}

} else if (isScalar(value)) {

elem.setTextContent(value.toString());

} else {

serializeObject(value, elem, doc);

}

parent.appendChild(elem);

}

}

private static boolean isScalar(Object x) {

return x instanceof String || x instanceof Number || x instanceof Boolean || x instanceof Character;

}

public static <T> T deserialize(Class<T> clazz, String filename) throws Exception {

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

DocumentBuilder builder = factory.newDocumentBuilder();

Document doc = builder.parse(new File(filename));

doc.getDocumentElement().normalize();

Element root = doc.getDocumentElement();

T instance = clazz.getDeclaredConstructor().newInstance();

deserializeObject(instance, root);

return instance;

}

private static void deserializeObject(Object obj, Element elem) throws Exception {

Class<?> clazz = obj.getClass();

NodeList children = elem.getChildNodes();

for (int i = 0; i < children.getLength(); i++) {

Node node = children.item(i);

if (node.getNodeType() != Node.ELEMENT\_NODE) continue;

Element child = (Element) node;

String fieldName = child.getNodeName();

Field field = null;

try {

field = clazz.getDeclaredField(fieldName);

} catch (NoSuchFieldException e) {

continue;

}

if (!field.isAnnotationPresent(XmlField.class)) continue;

field.setAccessible(true);

Class<?> fieldType = field.getType();

if (fieldType.isArray()) {

NodeList items = child.getElementsByTagName("item");

Class<?> componentType = fieldType.getComponentType();

Object array = Array.newInstance(componentType, items.getLength());

for (int j = 0; j < items.getLength(); j++) {

Element itemElem = (Element) items.item(j);

Object itemObj = componentType.getDeclaredConstructor().newInstance();

deserializeObject(itemObj, itemElem);

Array.set(array, j, itemObj);

}

field.set(obj, array);

} else if (fieldType == String.class) {

field.set(obj, child.getTextContent());

} else if (fieldType == int.class || fieldType == Integer.class) {

field.set(obj, Integer.parseInt(child.getTextContent()));

} else if (fieldType == boolean.class || fieldType == Boolean.class) {

field.set(obj, Boolean.parseBoolean(child.getTextContent()));

} else {

// Вложенный объект

Object nested = fieldType.getDeclaredConstructor().newInstance();

deserializeObject(nested, child);

field.set(obj, nested);

}

}

}

}

**ConsoleView.java**

package org.example.views;

import java.util.Scanner;

public class ConsoleView {

private final Scanner scanner = new Scanner(System.in);

public int showMenu() {

System.out.println("\nМеню:");

System.out.println("1 — Ввести данные и сохранить в XML");

System.out.println("2 — Загрузить из XML");

System.out.println("3 — Выход");

System.out.print("Ваш выбор: ");

while (!scanner.hasNextInt()) {

System.out.println("❌ Ошибка: введите число!");

scanner.next();

}

int choice = scanner.nextInt();

scanner.nextLine();

return choice;

}

public String readString(String prompt) {

String input;

do {

System.out.print(prompt);

input = scanner.nextLine();

if (input == null) input = "";

if (input.trim().isEmpty()) {

System.out.println("❌ Ошибка: поле не может быть пустым!");

}

} while (input.trim().isEmpty());

return input.trim();

}

public int readInt(String prompt) {

while (true) {

System.out.print(prompt);

String input = scanner.nextLine().trim();

if (input.isEmpty()) {

System.out.println("❌ Ошибка: введите число!");

continue;

}

try {

return Integer.parseInt(input);

} catch (NumberFormatException e) {

System.out.println("❌ Ошибка: введите корректное число!");

}

}

}

public void showMessage(String msg) {

System.out.println(msg);

}

}

**XmlSerializerServiceTest.java**

package org.example;

import org.example.models.Company;

import org.example.models.Person;

import org.example.models.Team;

import org.example.services.XmlSerializerService;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.io.TempDir;

import java.io.File;

import java.nio.file.Path;

import static org.junit.jupiter.api.Assertions.\*;

class XmlSerializerServiceTest {

@TempDir

Path tempDir;

@Test

void testSerializationAndDeserializationOnThreeClasses() throws Exception {

Person person1 = new Person("Анна", 28);

Person person2 = new Person("Иван", 32);

Team team = new Team("Разработка", new Person[]{person1, person2});

Company originalCompany = new Company("ТехноЛаб", team);

File xmlFile = tempDir.resolve("company.xml").toFile();

XmlSerializerService.serialize(originalCompany, xmlFile.getAbsolutePath());

Company restoredCompany = XmlSerializerService.deserialize(Company.class, xmlFile.getAbsolutePath());

assertEquals("ТехноЛаб", restoredCompany.getCompanyName());

assertEquals("Разработка", restoredCompany.getMainTeam().getTeamName());

assertEquals(2, restoredCompany.getMainTeam().getMembers().length);

Person restoredPerson1 = restoredCompany.getMainTeam().getMembers()[0];

assertEquals("Анна", restoredPerson1.getName());

assertEquals(28, restoredPerson1.getAge());

Person restoredPerson2 = restoredCompany.getMainTeam().getMembers()[1];

assertEquals("Иван", restoredPerson2.getName());

assertEquals(32, restoredPerson2.getAge());

}

}