МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ
Ордена трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский технический университет связи и информатики»

Кафедра Математическая кибернетика и информационные технологии

Отчет по лабораторной работе № 1

Выполнил: студент группы БПИ2401
Трухина Анастасия Александровна
Проверил: Харрасов Камиль Раисович

Москва,
2025

Оглавление

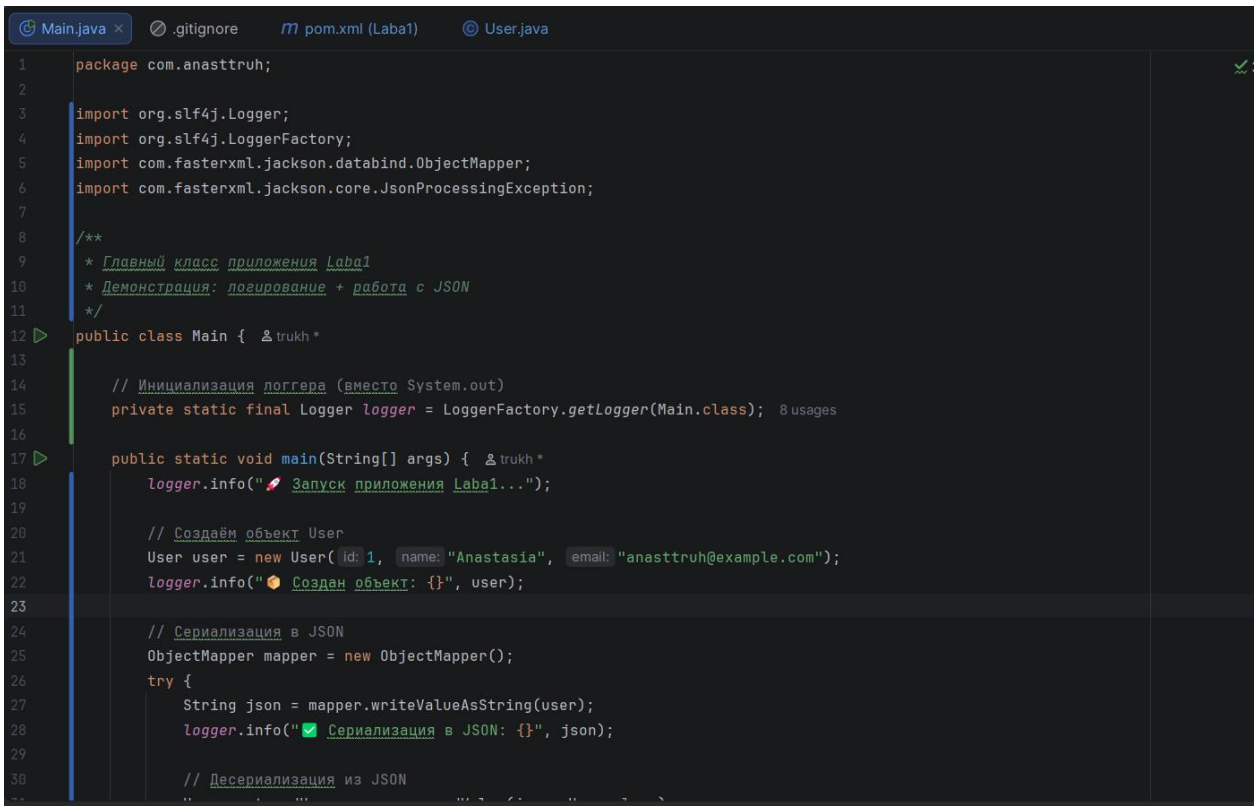
Цель работы:

Изучение принципов работы инструмента автоматизации сборки Apache Maven. Освоение управления зависимостями проекта, конфигурацией сборки через файл pom.xml, использование плагинов для компиляции, анализа кода и запуска приложений, а также понимание жизненного цикла сборки проекта.

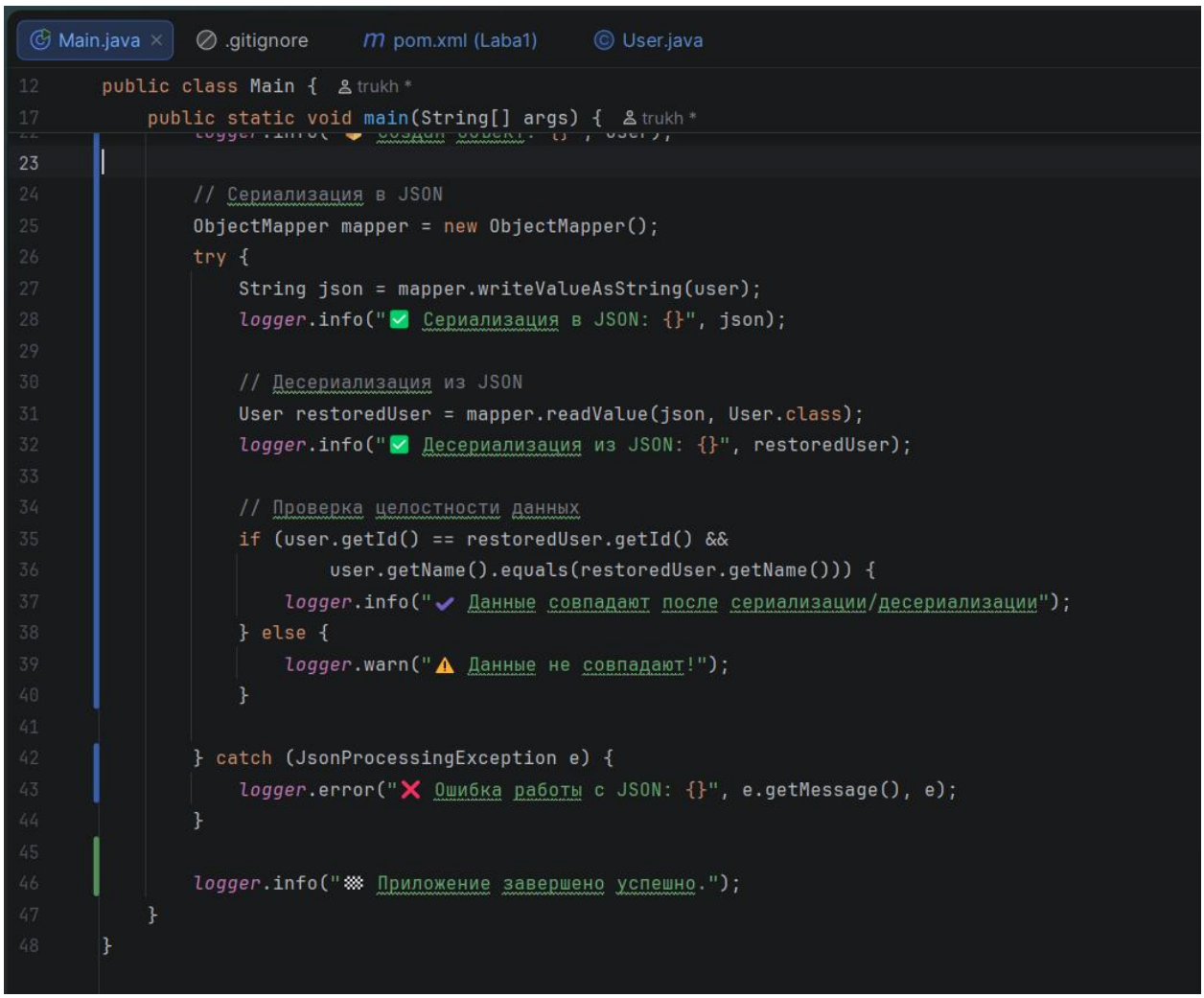
Задание:

1. Создать базовый Maven-проект в среде разработки и изучить его структуру.
2. Настроить файл pom.xml: указать координаты проекта (GAV), свойства компиляции и плагины.
3. Добавить и настроить плагин exec-maven-plugin для запуска основного класса приложения.
4. Внедрить зависимости для логирования (Log4j/SLF4J) и работы с JSON (Jackson), модифицировать код для их использования.
5. Изучить транзитивные зависимости проекта.
6. Подключить плагин статического анализа кода SpotBugs, выполнить проверку (mvn spotbugs:check) и устранить найденные проблемы.
7. Освоить команды сборки и запуска проекта через консоль (mvn clean install exec:java) и через среду разработки.

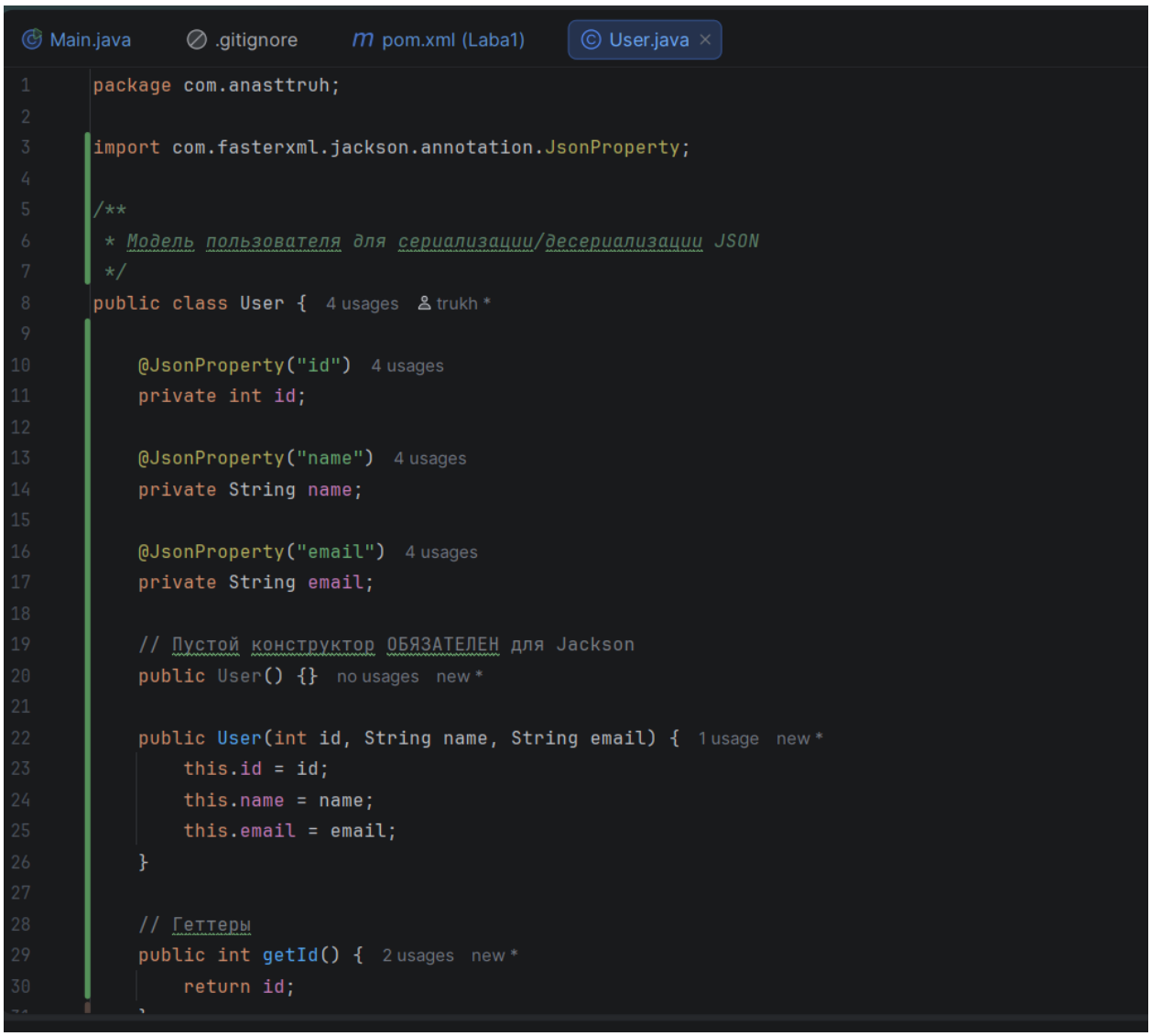
Код полученной программы:

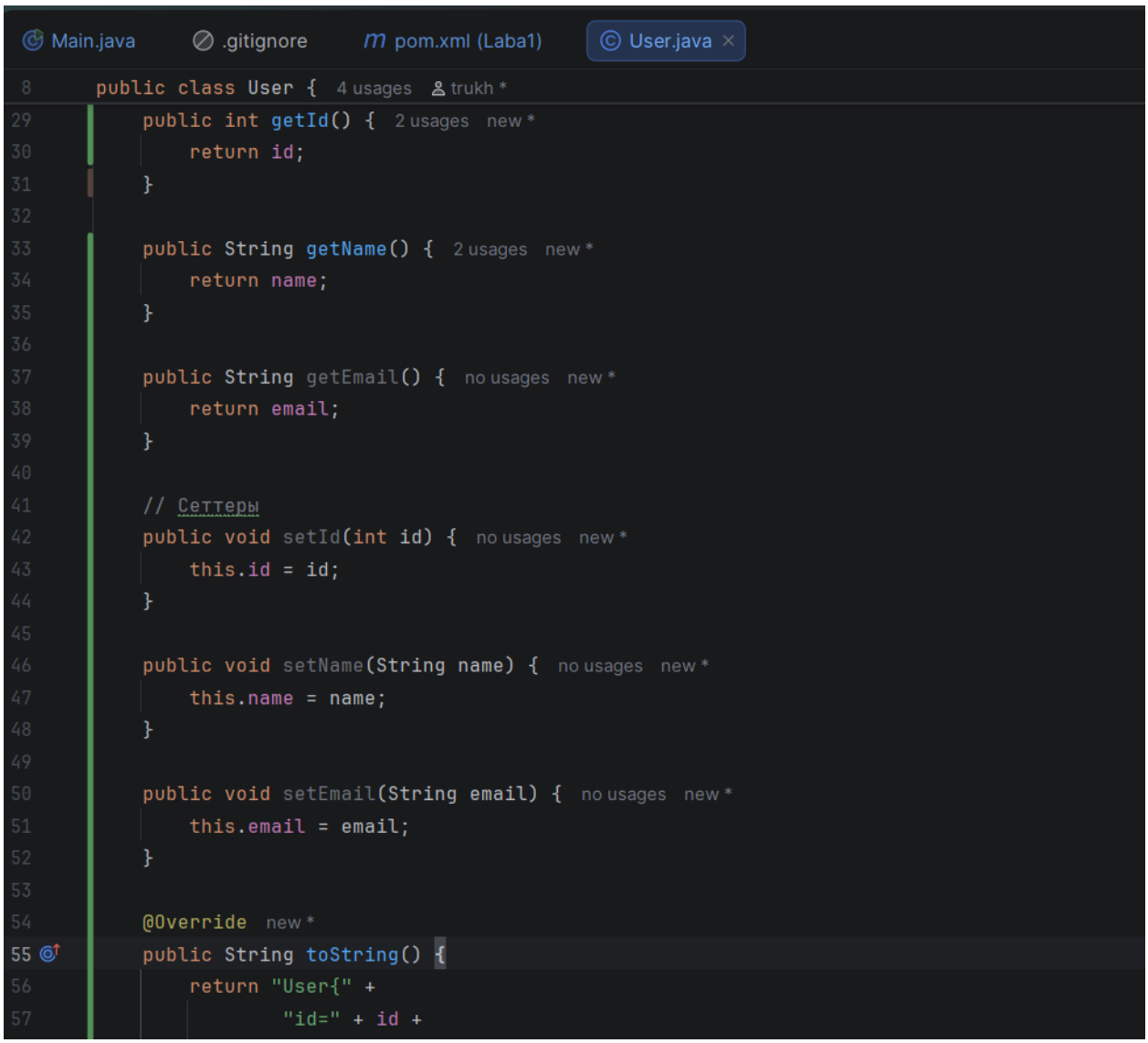


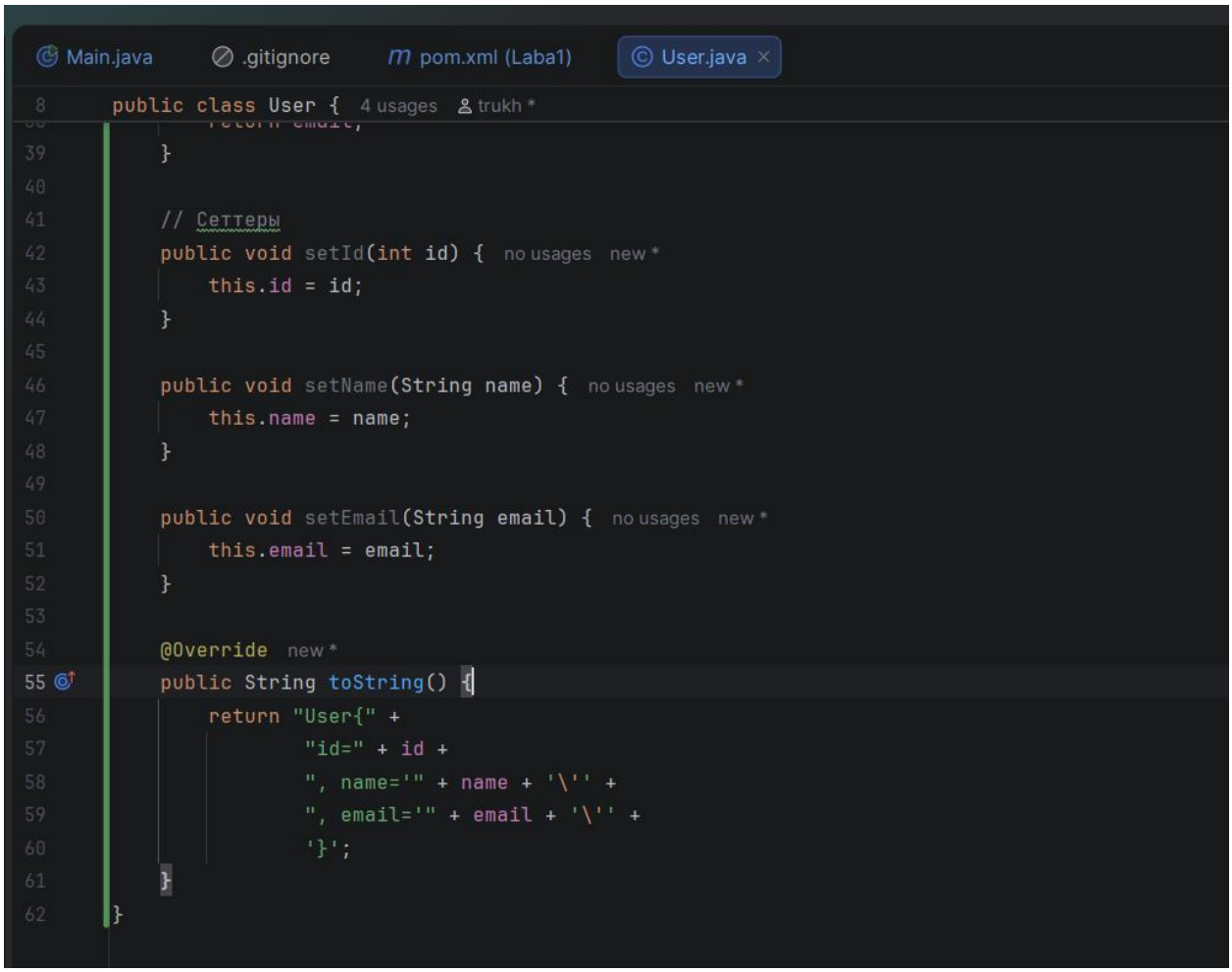
```java
package com.anasttruh;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.core.JsonProcessingException;

/**
 * Главный класс приложения Laba1
 * Демонстрация: логирование + работа с JSON
 */
public class Main {  trukh *

    // Инициализация логгера (вместо System.out)
    private static final Logger logger = LoggerFactory.getLogger(Main.class);  8 usages

    public static void main(String[] args) {  trukh *
        logger.info("🚀 Запуск приложения Laba1...");

        // Создаём объект User
        User user = new User( id: 1,  name: "Anastasia",  email: "anasttruh@example.com");
        logger.info("📦 Создан объект: {}", user);

        // Сериализация в JSON
        ObjectMapper mapper = new ObjectMapper();
        try {
            String json = mapper.writeValueAsString(user);
            logger.info("✅ Сериализация в JSON: {}", json);

            // Десериализация из JSON
```

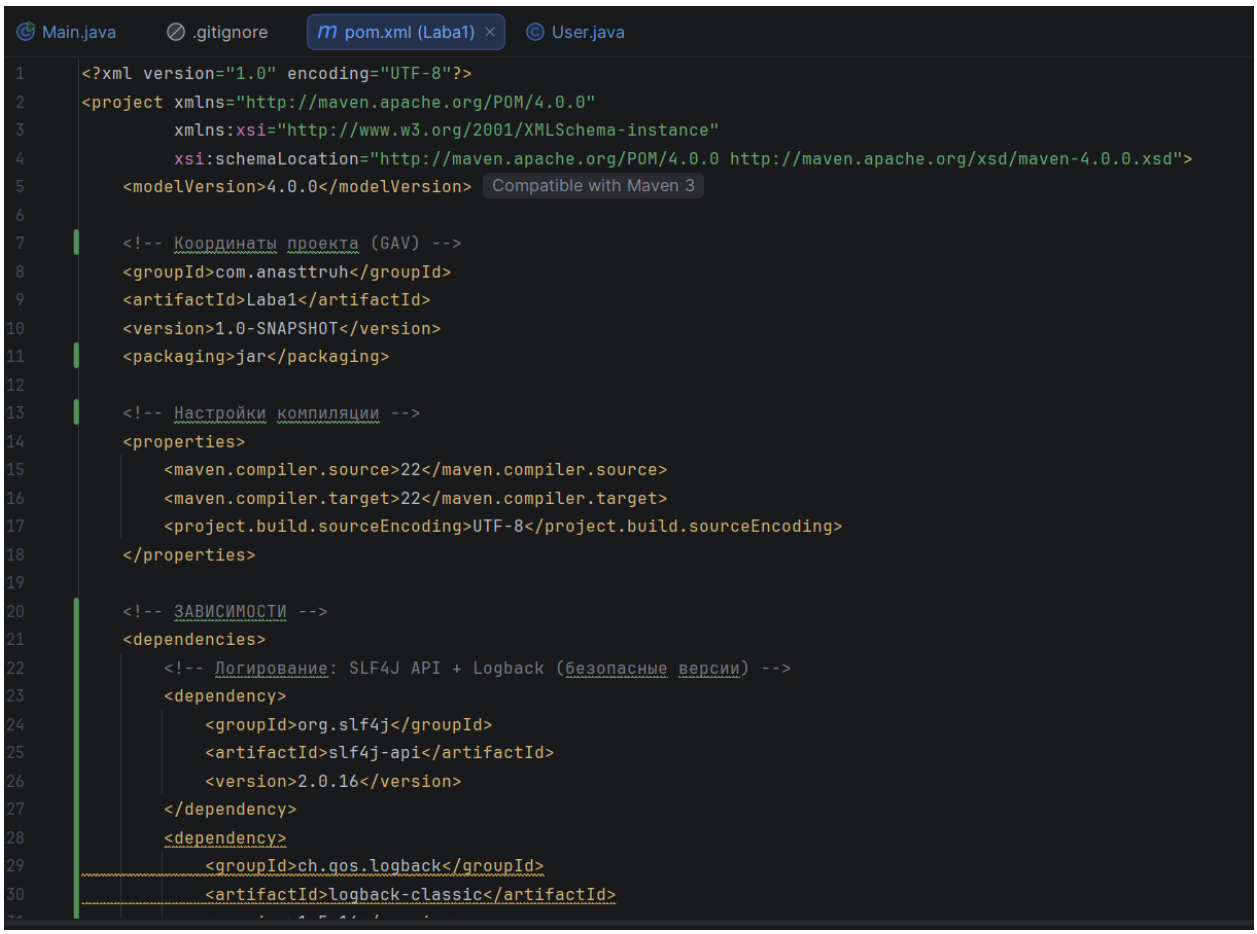


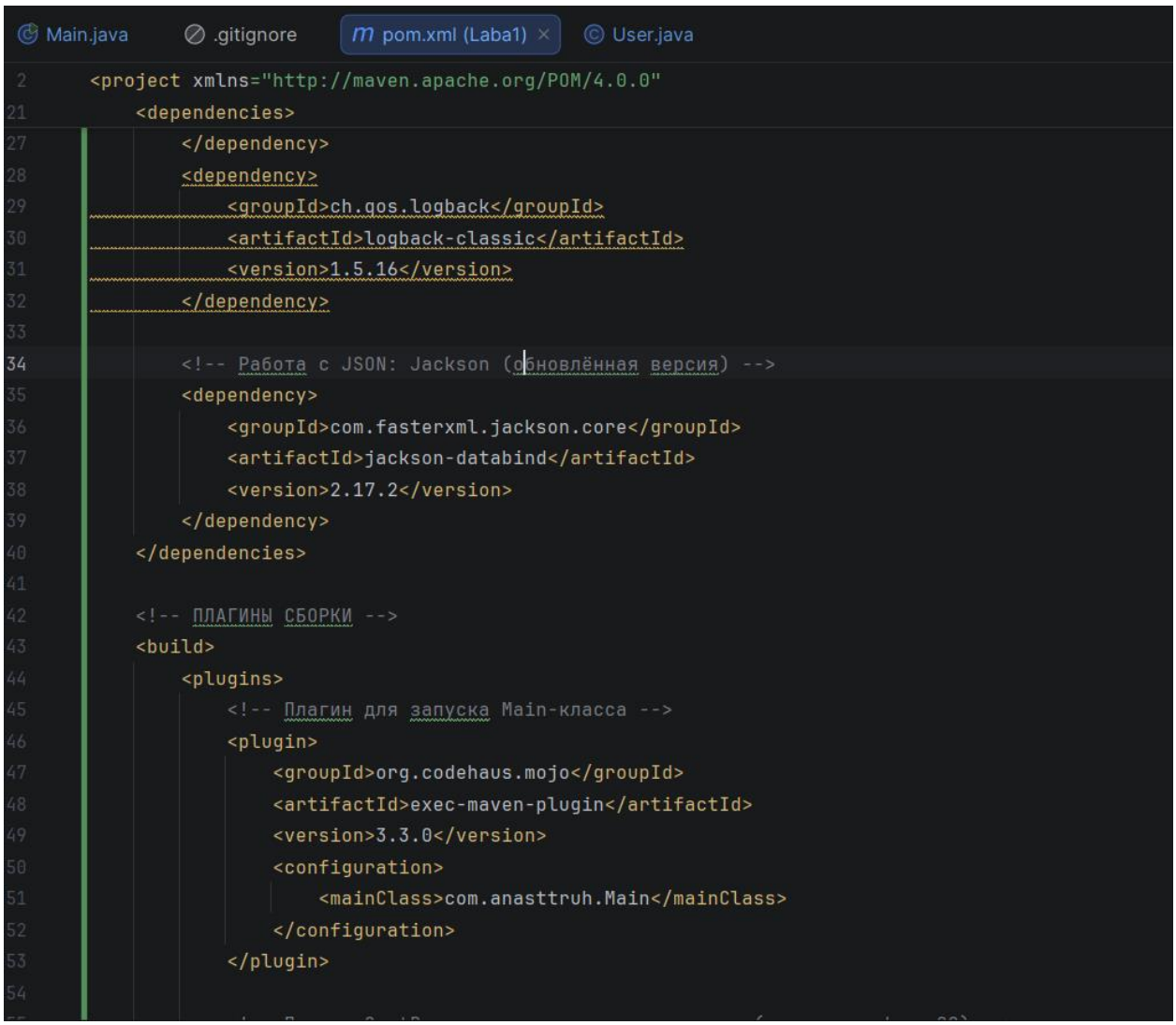
```java
public class Main {  trukh *
    public static void main(String[] args) {  trukh *

        // Сериализация в JSON
        ObjectMapper mapper = new ObjectMapper();
        try {
            String json = mapper.writeValueAsString(user);
            logger.info("✅ Сериализация в JSON: {}", json);

            // Десериализация из JSON
            User restoredUser = mapper.readValue(json, User.class);
            logger.info("✅ Десериализация из JSON: {}", restoredUser);

            // Проверка целостности данных
            if (user.getId() == restoredUser.getId() &&
                    user.getName().equals(restoredUser.getName())) {
                logger.info("✔ Данные совпадают после сериализации/десериализации");
            } else {
                logger.warn("⚠ Данные не совпадают!");
            }

        } catch (JsonProcessingException e) {
            logger.error("❌ Ошибка работы с JSON: {}", e.getMessage(), e);
        }

        logger.info("🏁 Приложение завершено успешно.");
    }
}
```

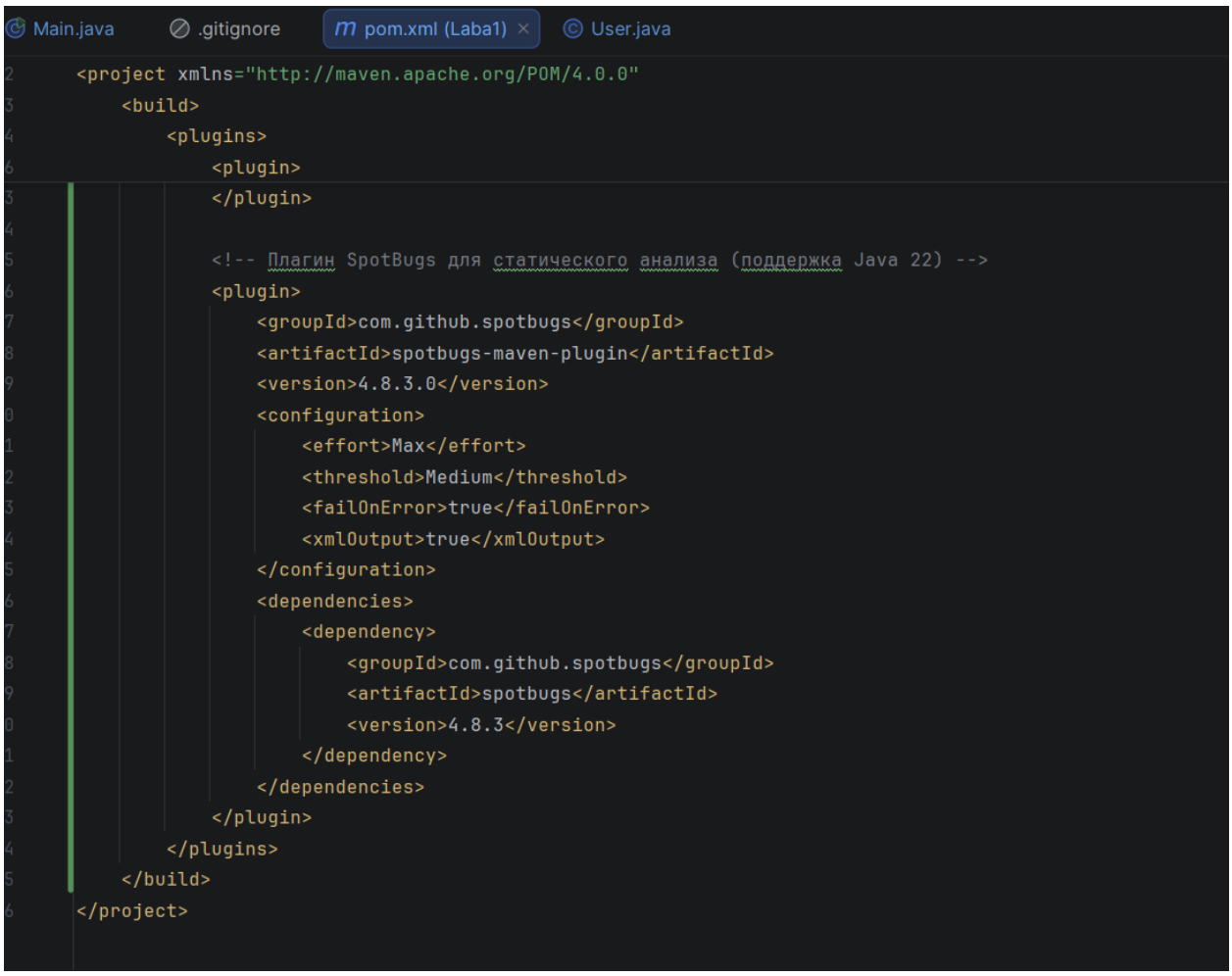

Main.java
.gitignore
pom.xml (Laba1)
User.java
package com.anasttruh;
import com.fasterxml.jackson.annotation.JsonProperty;
/**
* Модель пользователя для сериализации/десериализации JSON
*/
public class User { 4 usages trukh *
@JsonProperty("id") 4 usages
private int id;
@JsonProperty("name") 4 usages
private String name;
@JsonProperty("email") 4 usages
private String email;
// Пустой конструктор ОБЯЗАТЕЛЕН для Jackson
public User() {} no usages new *
public User(int id, String name, String email) { 1 usage new *
this.id = id;
this.name = name;
this.email = email;
}
// Геттеры
public int getId() { 2 usages new *
return id;


Main.java
.gitignore
pom.xml (Laba1)
User.java
public class User { 4 usages trukh *
public int getId() { 2 usages new *
return id;
}
public String getName() { 2 usages new *
return name;
}
public String getEmail() { no usages new *
return email;
}
// Сеттеры
public void setId(int id) { no usages new *
this.id = id;
}
public void setName(String name) { no usages new *
this.name = name;
}
public void setEmail(String email) { no usages new *
this.email = email;
}
@Override new *
public String toString() {
return "User{" +
"id=" + id +


Main.java
.gitignore
pom.xml (Laba1)
User.java
public class User { 4 usages trukh *
}
// Сеттеры
public void setId(int id) { no usages new *
this.id = id;
}
public void setName(String name) { no usages new *
this.name = name;
}
public void setEmail(String email) { no usages new *
this.email = email;
}
@Override new *
public String toString() {
return "User{" +
"id=" + id +
", name='" + name + '\'' +
", email='" + email + '\'' +
'}';
}
}


Main.java
.gitignore
pom.xml (Laba1)
User.java
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
Compatible with Maven 3
<!-- Координаты проекта (GAV) -->
<groupId>com.anasttruh</groupId>
<artifactId>Laba1</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>jar</packaging>
<!-- Настройки компиляции -->
<properties>
<maven.compiler.source>22</maven.compiler.source>
<maven.compiler.target>22</maven.compiler.target>
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>
<!-- ЗАВИСИМОСТИ -->
<dependencies>
<!-- Логирование: SLF4J API + Logback (безопасные версии) -->
<dependency>
<groupId>org.slf4j</groupId>
<artifactId>slf4j-api</artifactId>
<version>2.0.16</version>
</dependency>
<dependency>
<groupId>ch.qos.logback</groupId>
<artifactId>logback-classic</artifactId>


Main.java
.gitignore
pom.xml (Laba1)
User.java
<project xmlns="http://maven.apache.org/POM/4.0.0"
<dependencies>
</dependency>
<dependency>
<groupId>ch.qos.logback</groupId>
<artifactId>logback-classic</artifactId>
<version>1.5.16</version>
</dependency>
<!-- Работа с JSON: Jackson (обновлённая версия) -->
<dependency>
<groupId>com.fasterxml.jackson.core</groupId>
<artifactId>jackson-databind</artifactId>
<version>2.17.2</version>
</dependency>
</dependencies>
<!-- ПЛАГИНЫ СБОРКИ -->
<build>
<plugins>
<!-- Плагин для запуска Main-класса -->
<plugin>
<groupId>org.codehaus.mojo</groupId>
<artifactId>exec-maven-plugin</artifactId>
<version>3.3.0</version>
<configuration>
<mainClass>com.anasttruh.Main</mainClass>
</configuration>
</plugin>



Ответы на контрольные вопросы:

1. **Что такое Apache Maven и для чего он используется?** Apache Maven — это инструмент автоматизации сборки проектов на основе концепции POM (Project Object Model). Он используется для управления зависимостями, конфигурацией сборки и жизненным циклом проекта, упрощая поддержку и делая проекты переносимыми.

2. **Как установить Maven на различные операционные системы?** В системах на базе Linux (например, Ubuntu) можно использовать пакетный менеджер: sudo apt install maven. На других ОС установка производится путем скачивания дистрибутива с официального сайта и настройки переменных окружения (PATH, JAVA_HOME).

3. **Какова структура проекта Maven?** Базовая структура включает корневой файл pom.xml. Исходный код обычно располагается в src/main/java, ресурсы — в src/main/resources. Результаты компиляции сохраняются в директорию target.

4. **Что такое POM файл и какова его роль в проекте Maven?** POM (Project Object Model) — это XML-файл pom.xml в корне проекта. Он содержит всю информацию о проекте, необходимую для сборки: зависимости, плагины, цели и профили. Maven проверяет этот файл перед началом работы.

5. **Какова структура файла POM?** Основные элементы: корневой <project>, координаты (groupId, artifactId, version), <properties>, <dependencies> (зависимости), <build> (настройки сборки и плагины), <reporting> (отчеты).

6. **Что такое зависимости (dependencies) в Maven и как они определяются?** Зависимости — это библиотеки, необходимые проекту. Они определяются в секции <dependencies> через теги <dependency>, где указываются координаты: groupId, artifactId, version (GAV).

7. **Что такое репозиторий Maven и какие виды репозиториев существуют?** Репозиторий — это хранилище артефактов (jar-файлов). Существуют локальный репозиторий (на машине разработчика), центральный (public) и удаленные (корпоративные). В тексте упоминается локальный репозиторий, куда копируются файлы при фазе install.

8. **Как добавить зависимость в проект Maven?** Необходимо добавить блок <dependency> с указанием groupId, artifactId и version внутрь секции <dependencies> в файле pom.xml.

9. **Что такое плагины в Maven и как они используются?** Плагины — это расширения, выполняющие дополнительные действия при сборке (компиляция, проверка кода, упаковка). Они настраиваются в секции <build><plugins> или <reporting><plugins>.

10. **Как создать новый проект Maven с помощью команды Maven?** В тексте описано создание через среду разработки («Создать новый проект» -> Build System «Maven»). Через консоль обычно используется команда mvn archetype:generate, хотя в данном материале акцент сделан на IDE.

11. **Что такое цели (goals) и фазы (phases) в Maven и в чем их отличие?** Фазы — это этапы жизненного цикла сборки (например, clean, install, package). Цели — это конкретные задачи плагинов (например, exec:java). Команда может запускать фазы или конкретные цели плагинов.

12. **Как выполнить команду сборки проекта в Maven?** Через консоль командой mvn [фаза] [цель], например: mvn clean install exec:java. Также можно настроить конфигурацию запуска в среде разработки.

13. **Что такое жизненный цикл сборки (build lifecycle) в Maven?** Это последовательность фаз, через которые проходит проект при сборке (валидация, компиляция, тестирование, упаковка, установка). В тексте упоминаются фазы clean (очистка), install (сборка и установка в локальный репозиторий).

14. **Как настроить профили (profiles) в Maven для разных сред?** Профили позволяют активировать разные конфигурации (зависимости, плагины) для разных сред (разработка, продакшн). В pom.xml это секция <profiles>, хотя в данном тексте она подробно не раскрывается.

15. **Как управлять версиями зависимостей в Maven?** Версии указываются в теге <version> внутри зависимости. Также можно использовать свойства (<properties>) для централизованного управления версиями.

16. **Что такое "SNAPSHOT" версии в Maven и как они используются?** Версия с суффиксом -SNAPSHOT (например, 1.0-SNAPSHOT) обозначает, что проект находится в разработке, и результирующий jar-файл может меняться при каждой сборке.

17. **Как использовать Maven для создания отчета о качестве кода?** С помощью плагинов анализа кода, таких как maven-checkstyle-plugin, maven-pmd-plugin, findbugs-maven-plugin или SpotBugs, которые настраиваются в секции <reporting> или <build>.

18. **Какие команды Maven используются для очистки проекта, сборки, тестирования и установки?** Очистка: mvn clean. Сборка и установка: mvn install (включает компиляцию и тесты). Запуск: mvn exec:java.

19. **Как интегрировать Maven с системой контроля версий, такой как Git?** Файл pom.xml и исходный код добавляются в репозиторий Git. Директория target обычно исключается через .gitignore, так как она генерируется при сборке.

20. **Как добавить и настроить сторонний репозиторий в проекте Maven?** Сторонние репозитории добавляются в pom.xml в секцию <repositories> (в тексте подробно не описано, но это стандартная практика).

21. **Какие скоупы зависимостей существуют в Maven и для чего они используются?** Scope определяет видимость зависимости. Пример из текста: test (библиотека нужна только для тестов, например, junit). Другие стандартные: compile, provided, runtime.

22. **Чем отличается плагин от зависимости в Maven?** Зависимость — это библиотека, используемая кодом приложения во время работы. Плагин — это инструмент, используемый Maven во время сборки для выполнения задач (компиляция, проверка).

23. **Как работает транзитивность зависимостей?** Если проект зависит от библиотеки А, а библиотека А зависит от библиотеки Б, то проект автоматически получает библиотеку Б. В задании предлагается посмотреть появившиеся транзитивные зависимости при добавлении Jackson.

24. **Для чего нужен плагин surefire?** Плагин maven-surefire-plugin используется для запуска модульных тестов в фазе тестирования жизненного цикла Maven (в тексте упоминается выполнение тестов в фазе install).

25. **Как исключить транзитивную зависимость?** Внутри тега <dependency> используется секция <exclusions>, где указываются groupId и artifactId зависимости, которую нужно исключить.

Заключение

В ходе выполнения лабораторной работы был изучен инструмент Apache Maven, который стандартизирует процесс сборки и управления зависимостями Java-проектов. Была рассмотрена структура проекта и роль файла pom.xml (Project Object Model), в котором описываются координаты проекта, зависимости и настройки сборки. На практике были освоены навыки добавления внешних библиотек (логирование, JSON), настройки плагинов (compiler, exec, spotbugs) и управления жизненным циклом сборки. Использование Maven позволяет сделать проект более переносимым, упрощает подключение зависимостей и автоматизирует рутинные задачи, такие как компиляция, тестирование и анализ качества кода.

Ссылка на ГитХаб с файлами кода: NT-005-TN/ITiP-Lab-Works-Trukhina-4-Semestr