**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**



**МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**Факультет Информационных технологий**

***Кафедра Информатики и информационных технологий***

**направление подготовки**

**09.03.02 «Информационные системы и технологии»**

**ЛАБОРАТОРНАЯ РАБОТА № 5**

**Дисциплина:** технологии кроссплатформенного программирования

**Тема:** Документирование javadoc

**Выполнила: студентка группы 211-725**

Алюбаева Карина Ислямбековна

**Дата**  20.10.2023

**Проверил: \_\_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

**Дата, подпись** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  ***\_\_\_\_\_\_\_\_\_\_\_***

(Дата) (Подпись)

**Москва**

**2023**

**Оглавление**

[Ход работы 1](#_Toc148806518)

[Теоретическое обоснование 1](#_Toc148806519)

[Основная часть 3](#_Toc148806520)

[Задание 3](#_Toc148806521)

[Выполненное задание 1 3](#_Toc148806522)

[Выполненное задание 2 9](#_Toc148806523)

# Ход работы

## Теоретическое обоснование

При документировании приложения необходима также поддержка документации программы. Если документация и код разделены, то непроизвольно создаются сложности, связанные с необходимостью внесения изменений в соответствующие разделы сопроводительной документации при изменении программного кода.

Как правило, все существующие среды разработки IDE приложений Java предлагают решение по связыванию кода с документацией в процессе разработки с использованием javadoc. Для этого необходимо соответствующим образом написать комментарий к коду, т.е. документировать. Java комментарии необходимы как для комментирования программы, так и для составления или оформления документации.

Разработан специальный синтаксис для оформления документации в виде комментариев и инструмент для создания из комментариев документации. Этим инструментом является javadoc, который обрабатывая файл с исходным текстом программы, выделяет помеченную документацию из комментариев и связывает с именами соответствующих классов, методов и полей. Таким образом, при минимальных усилиях создания комментариев к коду, можно получить хорошую документацию к программе.

**javadoc** — это генератор документации в HTML-формате из комментариев исходного кода Java и определяет стандарт для документирования классов Java. Для создания доклетов и тэглетов, которые позволяют программисту анализировать структуру Javaприложения, javadoc также предоставляет API. В каждом случае комментарий должен находиться перед документируемым элементом.

При написании комментариев к кодам Java используют три типа комментариев:

// однострочный комментарий;

/\* многострочный комментарий \*/

/\*\* комментирование документации \*/

С помощью утилиты javadoc, входящей в состав JDK, комментарий документации можно извлекать и помещать в НТМL файл. Утилита javadoc позволяет вставлять HTML тэги и использовать специальные ярлыки (дескрипторы) документирования. НТМL тэги заголовков не используют, чтобы не нарушать стиль файла, сформированного утилитой.

Дескрипторы javadoc, начинающиеся со знака @, называются автономными и должны помещаться с начала строки комментария (лидирующий символ \* игнорируется). Дескрипторы, начинающиеся с фигурной скобки, например {@code}, называются встроенными и могут применяться внутри описания.

Комментарии документации применяют для документирования классов, интерфейсов, полей (переменных), конструкторов и методов. В каждом случае комментарий должен находиться перед документируемым элементом.

## Основная часть

### Задание

1. Прокомментировать программу из предыдущей работы.
2. С помощью Jvadoc создать HTML-файл, содержащий исходный код программы.

### Выполненное задание 1

Прокомментированный листинг:

import java.util.ArrayList;

/\*\*

\* представляет пример использования класса Conference

\* для подсчета школьников и студентов на конференции.

\* @author Карина

\*/

public class Main {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

// Создание экземпляров школьников и студентов

Schoolboy schoolboy1 = new Schoolboy("Маша", 8);

Schoolboy schoolboy2 = new Schoolboy("Петя", 7);

CollegeStudent student1 = new CollegeStudent("Егор", "Программист");

CollegeStudent student2 = new CollegeStudent("Марина", "Летчик");

// Создание конференции

Conference conference = new Conference();

// добавление участников

conference.addParticipant(schoolboy1);

conference.addParticipant(schoolboy2);

conference.addParticipant(student1);

conference.addParticipant(student2);

// Подсчет

int schoolboyCount = conference.countSchoolboys();

int collegeStudentCount = conference.countCollegeStudents();

System.out.println("Школьников: " + schoolboyCount);

System.out.println("Студентов: " + collegeStudentCount);

}

}

/\*\*

\* представляет базовый класс для студентов.

\* @author Карина

\*/

public class Student {

private String name; // Имя студента

/\*\*

\* Создает нового студента с указанным именем.

\*

\* @param name Имя студента.

\*/

public Student(String name) {

this.name = name;

}

/\*\*

\* Получить имя студента.

\*

\* @return Имя студента.

\*/

public String getName() {

return name;

}

}

/\*\*

\* представляет производный класс студентов, конкретно школьников.

\* @author Карина

\*/

class Schoolboy extends Student {

private int grade; // Класс школьника

/\*\*

\* Создает нового школьника с указанным именем и классом.

\*

\* @param name Имя школьника.

\* @param grade Класс школьника.

\*/

public Schoolboy(String name, int grade) {

super(name);

this.grade = grade;

}

/\*\*

\* Получить класс школьника.

\*

\* @return Класс школьника.

\*/

public int getGrade() {

return grade;

}

}

/\*\*

\* представляет производный класс студентов, конкретно студентов колледжа.

\* @author Карина

\*/

class CollegeStudent extends Student {

private String major; // Специальность студента

/\*\*

\* Создает нового студента колледжа с указанным именем и специальностью.

\*

\* @param name Имя студента.

\* @param major Специальность студента.

\*/

public CollegeStudent(String name, String major) {

super(name);

this.major = major;

}

/\*\*

\* Получить специальность студента колледжа.

\*

\* @return Специальность студента.

\*/

public String getMajor() {

return major;

}

}

/\*\*

\* представляет собой класс, который может содержать как школьников, так и студентов колледжа.

\* @author Карина

\*/

class Conference {

private ArrayList<Student> participants = new ArrayList<>();

/\*\*

\* Добавить студента (школьника или студента колледжа) на конференцию.

\*

\* @param student Студент, который будет добавлен на конференцию.

\*/

public void addParticipant(Student student) {

participants.add(student);

}

/\*\*

\* Подсчитать количество школьников на конференции.

\*

\* @return Количество школьников на конференции.

\*/

public int countSchoolboys() {

int count = 0;

for (Student participant : participants) {

if (participant instanceof Schoolboy) {

count++;

}

}

return count;

}

/\*\*

\* Подсчитать количество студентов колледжа на конференции.

\*

\* @return Количество студентов колледжа на конференции.

\*/

public int countCollegeStudents() {

int count = 0;

for (Student participant : participants) {

if (participant instanceof CollegeStudent) {

count++;

}

}

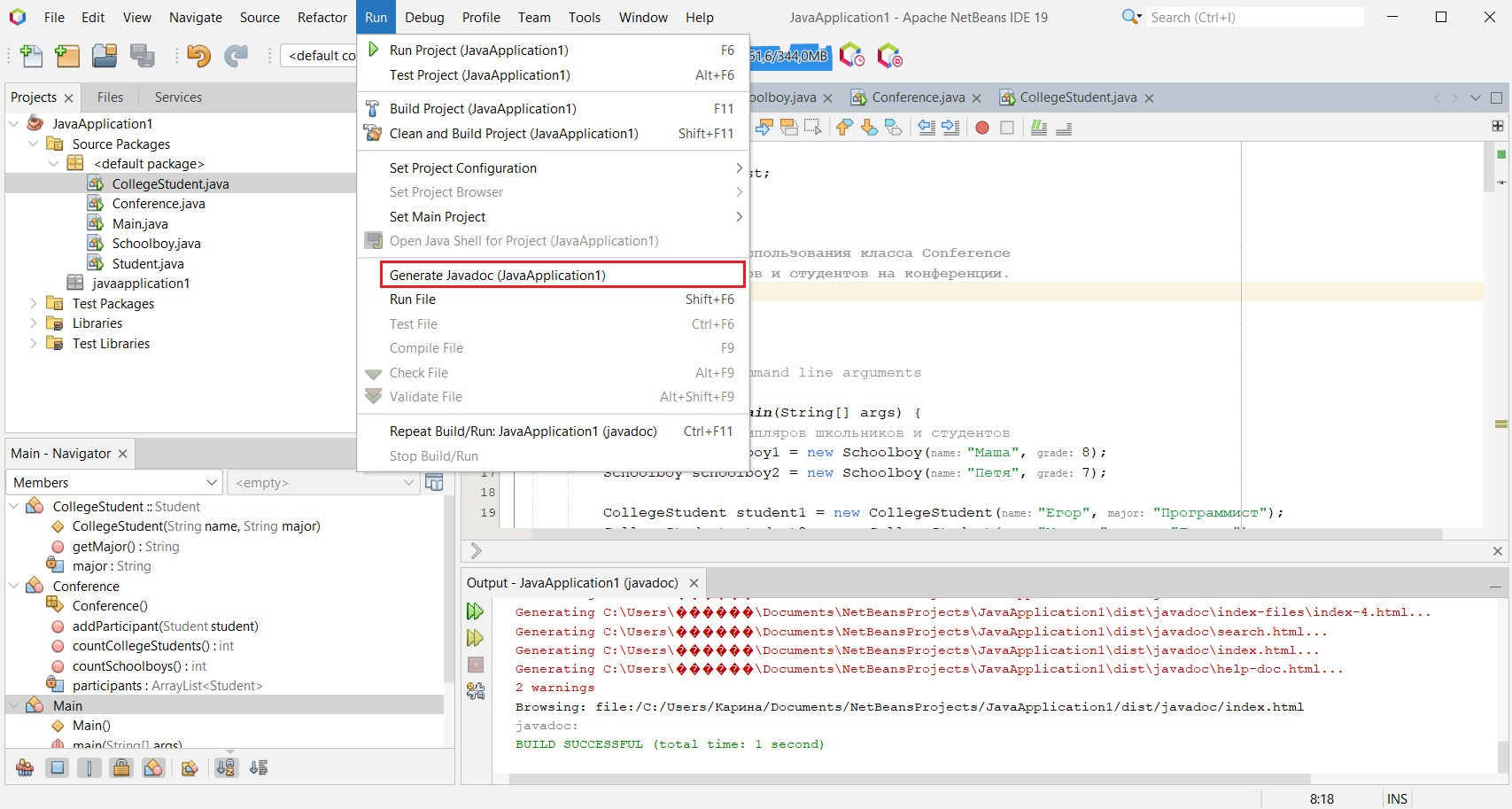
return count;

}

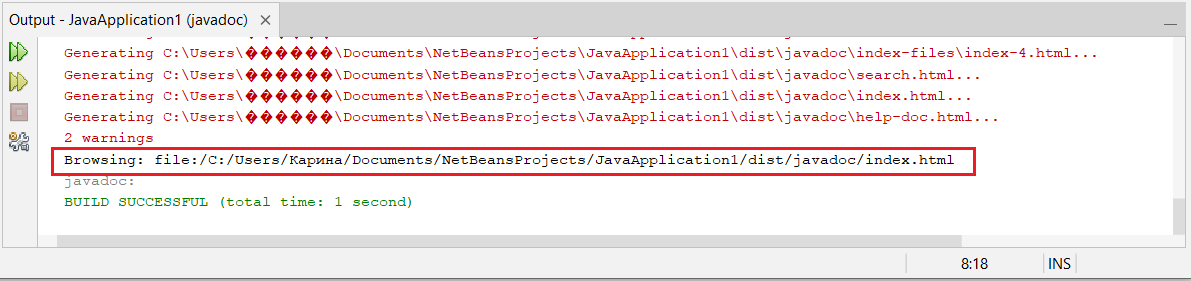
}

### Выполненное задание 2

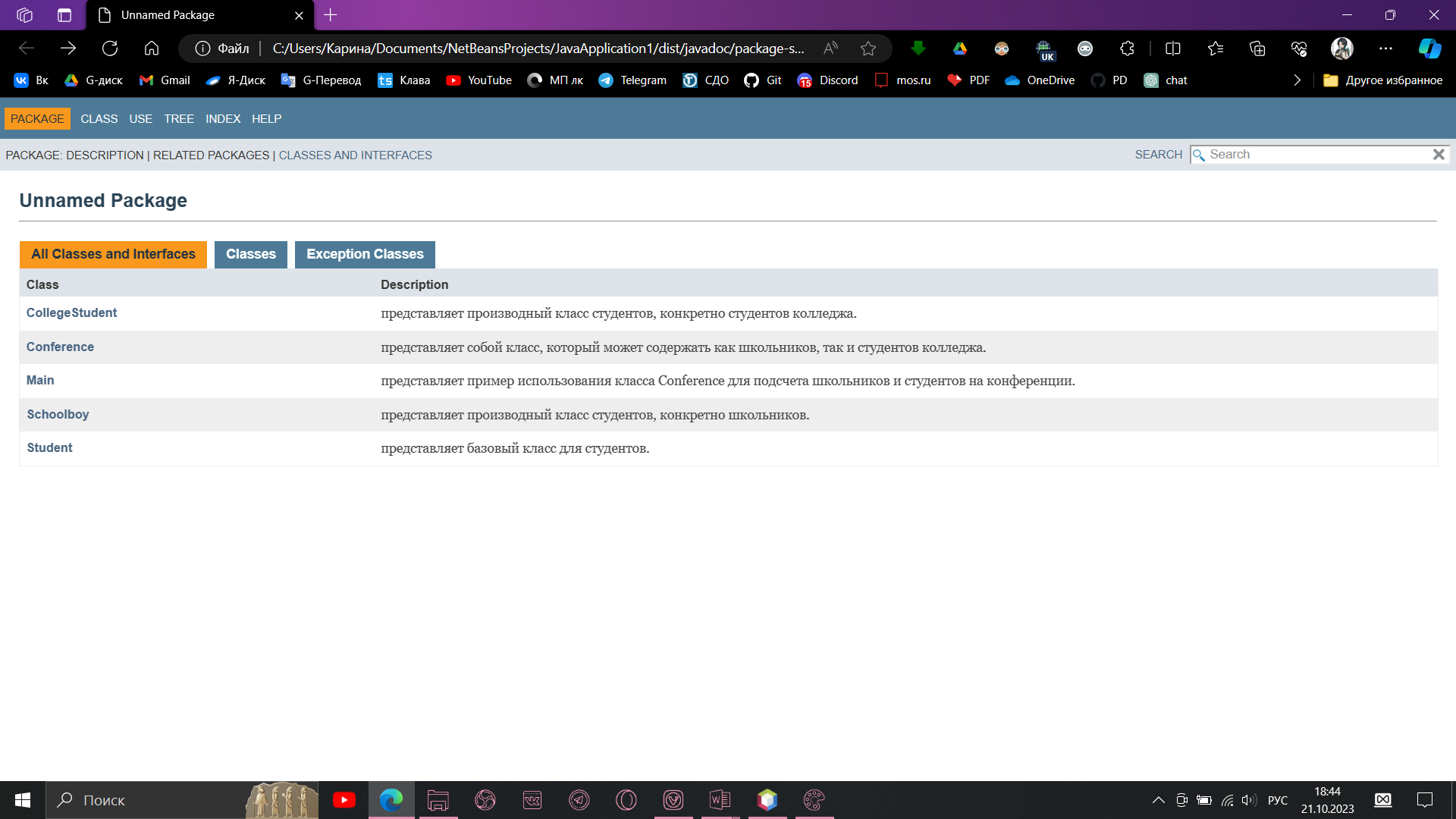
Run 🡪 Generate Javadoc



В консоли появится результат генерации документа и путь, по которому его можно найти.



Окно браузера с открытым html-файлом:



Можно отдельно посмотреть классы, классы-наследники. Если нажать, на класс, то появится подробная информация об этом классе.

