**МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Программное обеспечение информационных технологий»

**Практика написания программного кода**

**ЛАБОРАТОРНАЯ РАБОТА № 2.**

Язык моделирования UML. Построение диаграмм классов

Выполнил:

**Шведов Н.В.**

Проверил:

**Мищенко И.И.**

Могилев 2025

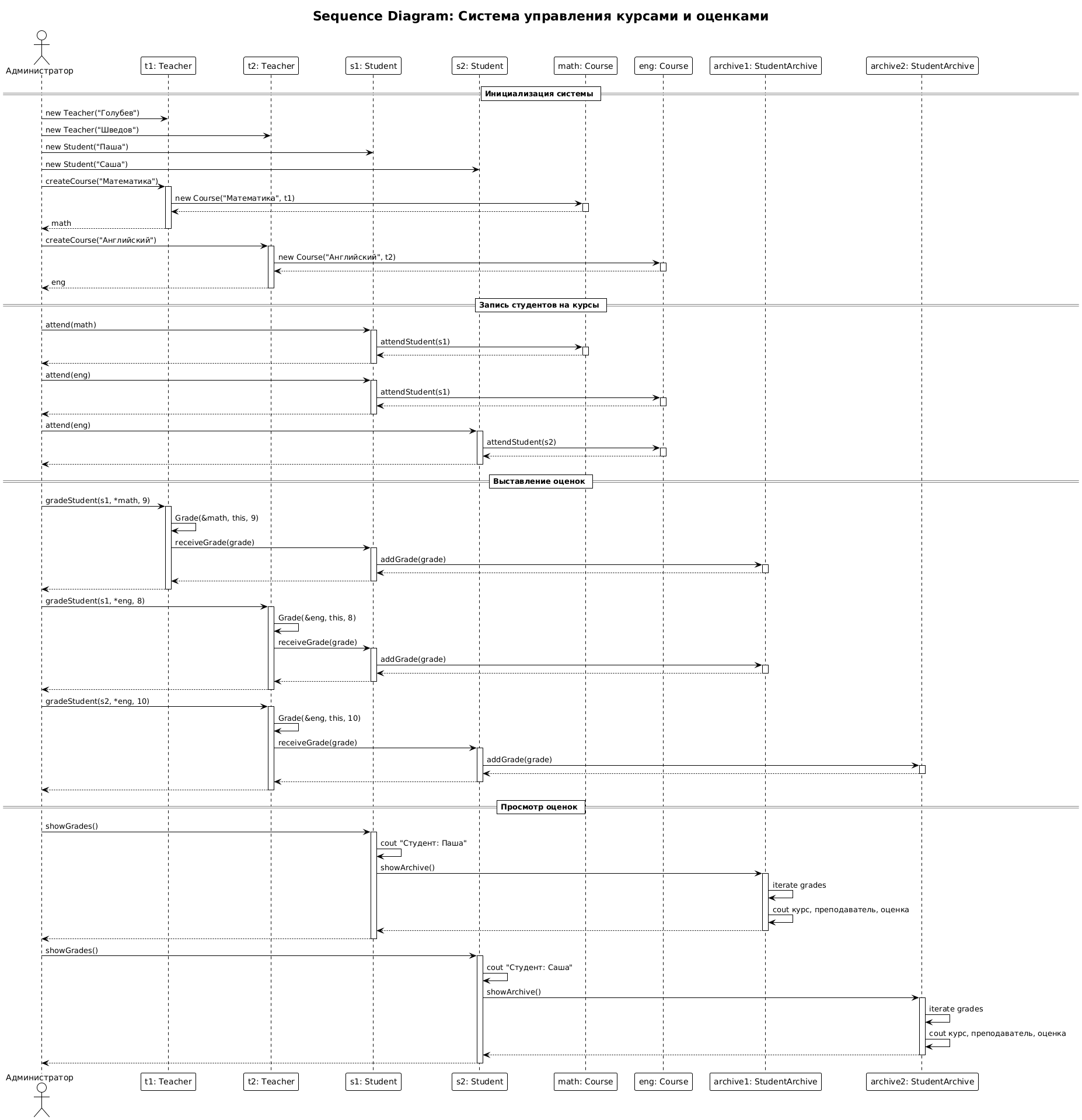
**ЛАБОРАТОРНАЯ РАБОТА № 2.**

Язык моделирования UML. Построение диаграмм взаимодействия

***Цель работы***: получить навыки построения диаграмм взаимодействия на языке моделирования UML.

**Задание 1**

Нарисовать UML-диаграмму взаимодействия объектов программной системы. Система Факультатив. Преподаватель объявляет запись на Курс. Студент записывается на Курс, обучается и по окончании Преподаватель выставляет Оценку, которая сохраняется в Архиве Студентов. Преподавателей и Курсов при обучении может быть несколько.



Реализация UML диаграммы на языке программирования C++:

main.cpp

#include <iostream>

#include <vector>

#include <string>

#include <map>

#include <memory>

class Course;

class Teacher;

class Student;

class INamed {

public:

virtual ~INamed() = default;

virtual std::string getName() const = 0;

};

class Grade {

public:

const Course\* course;

const Teacher\* teacher;

int value;

Grade(const Course\* course, const Teacher\* teacher, int val)

: course(course), teacher(teacher), value(val) {

}

};

class StudentArchive {

std::vector<Grade> grades;

public:

void addGrade(const Grade& grade) {

grades.push\_back(grade);

}

void showArchive() const;

};

class Teacher : public INamed {

std::string name;

public:

Teacher(const std::string& name) : name(name) {}

std::shared\_ptr<Course> createCourse(const std::string& title);

void gradeStudent(Student& student, const Course& course, int value);

std::string getName() const override { return name; }

};

class Course : public INamed {

std::string title;

Teacher\* teacher;

std::vector<Student\*> attendedStudents;

public:

Course(const std::string& title, Teacher\* teacher)

: title(title), teacher(teacher) {

}

void attendStudent(Student\* student) {

attendedStudents.push\_back(student);

}

std::string getName() const override { return title; }

std::string getTitle() const { return title; }

Teacher\* getTeacher() const { return teacher; }

};

class Student : public INamed {

std::string name;

StudentArchive archive;

std::vector<std::shared\_ptr<Course>> attendedCourses;

public:

Student(const std::string& name) : name(name) {}

void attend(std::shared\_ptr<Course> course);

void receiveGrade(const Grade& grade) {

archive.addGrade(grade);

}

void showGrades() const {

std::cout << "\n" << "Студент: " << name;

archive.showArchive();

}

std::string getName() const override { return name; }

};

void StudentArchive::showArchive() const {

std::cout << ", архив оценок:\n";

for (const auto& grade : grades) {

std::cout << "курс - " << grade.course->getName()

<< ", преподаватель - " << grade.teacher->getName()

<< ", оценка - " << grade.value << "\n" ;

}

}

void Student::attend(std::shared\_ptr<Course> course) {

attendedCourses.push\_back(course);

course->attendStudent(this);

}

std::shared\_ptr<Course> Teacher::createCourse(const std::string& title) {

return std::make\_shared<Course>(title, this);

}

void Teacher::gradeStudent(Student& student, const Course& course, int value) {

student.receiveGrade(Grade(&course, this, value));

}

int main() {

std::locale::global(std::locale(""));

Teacher t1("Голубев");

Teacher t2("Шведов");

Student s1("Паша");

Student s2("Саша");

auto math = t1.createCourse("Математика");

auto eng = t2.createCourse("Английский");

s1.attend(math);

s1.attend(eng);

s2.attend(eng);

t1.gradeStudent(s1, \*math, 9);

t2.gradeStudent(s1, \*eng, 8);

t2.gradeStudent(s2, \*eng, 10);

s1.showGrades();

s2.showGrades();

return 0;

}

Вывод программы

