ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА»  
(РУТ (МИИТ))

Институт транспортной техники и систем управления

Кафедра «Управление и защита информации»

ОТЧЁТ  
О ПРАКТИЧЕСКОЙ РАБОТЕ № 2

По дисциплине «Языки программирования»

Выполнил: ст. гр. ТКИ – 241

Байло А.В.

Проверил: к.т.н., доц.

Васильева М. А.

Москва 2023

**Задание:** Написать архитектуру Университета, содержащего классы группа, студент, преподаватель, дисциплина, с использованием Smart-pointers

**Код программы:**

#include "Teacher.h"

Teacher::Teacher(std::string name, std::string graduate)

:teacherName(name) , graduate(graduate), subject(nullptr)

{

}

std::shared\_ptr<Subject> Teacher::getSubject()

{

return this->subject;

}

void Teacher::setSubject(std::shared\_ptr<Subject> subject)

{

this->subject = subject;

}

void Teacher::setGroup(std::shared\_ptr<Group> group)

{

this->groups.emplace\_back(group);

}

std::ostream& operator<<(std::ostream& os, const Teacher& teacher)

{

return os << teacher.teacherName;

}

#pragma once

#include <string>

#include <iostream>

#include <memory>

#include <vector>

class Subject;

class Group;

/\*\*

\* @brief class Teacher

\*

\*/

class Teacher

{

public:

/\*\*

\* @brief Construct a new Teacher object

\*

\* @param name teacher's name

\* @param graduate teacher's graduate

\*/

Teacher(std::string name, std::string graduate);

/\*\*

\* @brief Get the Subject object taught by the teacher

\*

\* @return std::shared\_ptr<Subject> a pointer to subject object

\*/

std::shared\_ptr<Subject> getSubject();

/\*\*

\* @brief Set the Subject object taught by the teacher

\*

\* @param subject pointer to Subject object to teach

\*/

void setSubject(std::shared\_ptr<Subject> subject);

/\*\*

\* @brief Set the Group object taught by the teacher

\*

\* @param group pointer to group object to teach

\*/

void setGroup(std::shared\_ptr<Group> group);

/\*\*

\* @brief overload << operator

\*

\* @param os output stream

\* @param teacher teacher object to put into stream

\* @return std::ostream& stream which contains teacher object

\*/

friend std::ostream& operator<<(std::ostream& os, const Teacher& teacher);

private:

std::shared\_ptr<Subject> subject;

const std::string teacherName;

std::string graduate;

std::vector<std::shared\_ptr<Group>> groups;

};

#include "group.h"

#include <cinttypes>

#include <initializer\_list>

#include <iostream>

#include <memory>

#include <string>

#include <vector>

Group::Group(std::string groupName)

:groupName(groupName)

{

}

void Group::addStudent(std::shared\_ptr<Student>& student)

{

students.emplace\_back(student);

student->setGroup(shared\_from\_this());

}

void Group::setSubject(std::shared\_ptr<Subject> subject, std::shared\_ptr<Teacher> teacher)

{

subjects.emplace\_back(subject);

teacher->setGroup(shared\_from\_this());

}

std::ostream& operator<<(std::ostream& os,const Group& group)

{

os << group.groupName << "\n";

for(auto it = group.students.cbegin(); it < group.students.cend(); it++)

{

if(auto student = it->lock())

{

os << \*it->lock() << "\n";

}

}

return os;

}

void Group::deleteStudent(std::shared\_ptr<Student>& student)

{

for (auto it = students.cbegin();it < students.cend(); it++)

{

if (it->lock() == student)

{

students.erase(it);

}

}

student->setGroup(nullptr);

}

std::string Group::toString()

{

return groupName;

}

#include "student.h"

#include <initializer\_list>

#include <memory>

#include <string>

#include <algorithm>

#include "subject.h"

#pragma once

class Subject;

class Student;

/\*\*

\* @brief class Group

\*

\*/

class Group

: public std::enable\_shared\_from\_this<Group>

{

public:

/\*\*

\* @brief Construct a new Group object

\*

\* @param groupName group's name

\*/

Group(std::string groupName);

/\*\*

\* @brief function which adds student to the group

\*

\* @param student pointer to student object to add

\*/

void addStudent(std::shared\_ptr<Student>& student);

/\*\*

\* @brief function which deletes student from group

\*

\* @param student pointer to student object to delete

\*/

void deleteStudent(std::shared\_ptr<Student>& student);

/\*\*

\* @brief Set the Subject object to the group

\*

\* @param subject pointer to subject object which to set

\* @param teacher pointer to teacher object whict to set

\*/

void setSubject(std::shared\_ptr<Subject> subject, std::shared\_ptr<Teacher> teacher);

/\*\*

\* @brief function which convert group object to std::string

\*

\* @return std::string group object

\*/

std::string toString();

/\*\*

\* @brief overload << operator

\*

\* @param os output stream

\* @param group group object to put into stream

\* @return std::ostream& stream which contains group object

\*/

friend std::ostream& operator<<(std::ostream& os,const Group& group);

private:

std::vector<std::weak\_ptr<Student>> students;

std::string groupName;

std::vector<std::weak\_ptr<Subject>> subjects;

};

#include "student.h"

#include <exception>

Student::Student(std::string studentName,size\_t studentAge, size\_t recordBookNumber)

:studentName(studentName) , age(studentAge) , recordBookNumber(recordBookNumber) , group(nullptr)

{

}

void Student::setGroup(std::shared\_ptr<Group> group)

{

this->group = group;

}

std::ostream& operator<<(std::ostream& os, const Student& Student)

{

os << Student.studentName << "\t" << Student.age << "\t" << Student.recordBookNumber;

return os;

}

std::shared\_ptr<Group> Student::getGroup()

{

return group;

}

#pragma once

#include <iostream>

#include <memory>

#include <string>

#include <vector>

class Group;

/\*\*

\* @brief class Student

\*

\*/

class Student

{

public:

/\*\*

\* @brief Construct a new Student object

\*

\* @param studentName student's name

\* @param studentAge student's age

\* @param recordBookNumber students's recordBookNumber

\*/

Student(std::string studentName,size\_t studentAge, size\_t recordBookNumber);

/\*\*

\* @brief Set the Group object to student

\*

\* @param group pointer to the group to which the student belongs

\*/

void setGroup(std::shared\_ptr<Group> group);

/\*\*

\* @brief Get the Group object

\*

\* @return std::shared\_ptr<Group> a pointer to the group to which the student belongs

\*/

std::shared\_ptr<Group> getGroup();

/\*\*

\* @brief overload << operator

\*

\* @param os output stream

\* @param student student object to put into stream

\* @return std::ostream& stream which contains student object

\*/

friend std::ostream& operator<<(std::ostream& os, const Student& student);

private:

std::shared\_ptr<Group> group;

std::string studentName;

size\_t age;

size\_t recordBookNumber;

};

#include "subject.h"

Subject::Subject(std::string subjectName)

:subjectName(subjectName)

{

}

void Subject::setTeacher(std::shared\_ptr<Teacher>& teacher)

{

subjectTeachers.emplace\_back(teacher);

teacher->setSubject(shared\_from\_this());

}

void Subject::deleteTeacher(std::shared\_ptr<Teacher>& teacher)

{

for (auto it = subjectTeachers.cbegin();it < subjectTeachers.cend(); it++)

{

if (it->lock() == teacher)

{

subjectTeachers.erase(it);

}

}

teacher->setSubject(nullptr);

}

std::ostream& operator<<(std::ostream& os,const Subject& subject)

{

os << subject.subjectName << "\n";

for(auto it = subject.subjectTeachers.cbegin(); it < subject.subjectTeachers.cend(); it++)

{

if(auto teacher = it->lock())

{

os << \*it->lock() << "\n";

}

}

return os;

}

#include "Teacher.h"

#include <memory>

#include <vector>

#pragma once

class Teacher;

/\*\*

\* @brief class Subject

\*

\*/

class Subject:public std::enable\_shared\_from\_this<Subject>

{

public:

/\*\*

\* @brief Construct a new Subject object by its name

\*

\* @param subjectName name of the subject

\*/

Subject(std::string subjectName);

/\*\*

\* @brief overload << operator

\*

\* @param os output stream

\* @param subject subject object to put into stream

\* @return std::ostream& stream which contains subject object

\*/

friend std::ostream& operator<<(std::ostream& os,const Subject& subject);

/\*\*

\* @brief Set teacher to subject

\*

\* @param teacher teacher which teaches Subject

\*/

void setTeacher(std::shared\_ptr<Teacher>& teacher);

/\*\*

\* @brief delete Teacher from Subject (Subject may cantains more than 1 teacher)

\*

\* @param teacher a pointer to Teacher object to delete

\*/

void deleteTeacher(std::shared\_ptr<Teacher>& teacher);

private:

std::vector<std::weak\_ptr<Teacher>> subjectTeachers;

const std::string subjectName;

};

**Тесты:**

#include <cstddef>

#include <iostream>

#include "group.h"

#include "group.cpp"

#include "student.h"

#include "student.cpp"

#include "subject.h"

#include "subject.cpp"

#include "Teacher.h"

#include "Teacher.cpp"

#include <gtest/gtest.h>

#include <memory>

class Student;

class Teacher;

class Subject;

class Group;

/\*\*

\* Tests consist of 3 parts

1) create a object and use some method

2) create actual and expected values

3) ASSERT

\*

\*/

TEST(GroupMetodsTest, addStudentTets)

{

Group defaultGroup ("TKI");

Student defaultStudent("Ivan" , 20 , 1234);

std::shared\_ptr<Student> defaultSharedStudent = std::make\_shared<Student>(defaultStudent);

std::shared\_ptr<Group> defaultSharedGroup = std::make\_shared<Group>(defaultGroup);

defaultSharedGroup->addStudent(defaultSharedStudent);

std::shared\_ptr<Group> actual = defaultSharedGroup;

std::shared\_ptr<Group> expected = defaultSharedStudent->getGroup();

ASSERT\_EQ(actual, expected);

}

TEST(GroupMetodsTest, deleteStudentTest)

{

Group defaultGroup ("TKI");

Student defaultStudent("Ivan" , 20 , 1234);

std::shared\_ptr<Student> defaultSharedStudent = std::make\_shared<Student>(defaultStudent);

std::shared\_ptr<Group> defaultSharedGroup = std::make\_shared<Group>(defaultGroup);

defaultSharedGroup->addStudent(defaultSharedStudent);

defaultSharedGroup->deleteStudent(defaultSharedStudent);

std::shared\_ptr<Group> actual = defaultSharedStudent->getGroup();

std::shared\_ptr<Group> expected = nullptr;

std::string actualStr = defaultSharedGroup->toString();

std::string expectedStr = "TKI";

ASSERT\_EQ(actual, expected);

ASSERT\_EQ(actualStr, expectedStr);

}

TEST(GroupMetodsTest, setSubjectTest)

{

Teacher defaultTeacher("IvanIvanovich", "Doctor");

Group defaultGroup ("TKI");

Subject defaultSubject("Math");

std::shared\_ptr<Teacher> defaultSharedTeacher = std::make\_shared<Teacher>(defaultTeacher);

std::shared\_ptr<Subject> defaultSharedSubject = std::make\_shared<Subject>(defaultSubject);

std::shared\_ptr<Group> defaultSharedGroup = std::make\_shared<Group>(defaultGroup);

defaultSharedSubject->setTeacher(defaultSharedTeacher);

defaultSharedGroup->setSubject(defaultSharedSubject, defaultSharedTeacher);

std::shared\_ptr<Group> actual = defaultSharedGroup;

std::shared\_ptr<Group> expected = defaultSharedGroup;

ASSERT\_EQ(actual, expected);

}

TEST(SubjectMetodsTest, setTeacherTest)

{

Teacher defaultTeacher("IvanIvanovich", "Doctor");

Subject defaultSubject("Math");

std::shared\_ptr<Teacher> defaultSharedTeacher = std::make\_shared<Teacher>(defaultTeacher);

std::shared\_ptr<Subject> defaultSharedSubject = std::make\_shared<Subject>(defaultSubject);

defaultSharedSubject->setTeacher(defaultSharedTeacher);

std::shared\_ptr<Subject> actual = defaultSharedTeacher->getSubject();

std::shared\_ptr<Subject> expected = defaultSharedSubject;

ASSERT\_EQ(actual, expected);

}

TEST(SubjectMetodsTest, deleteTeacherTest)

{

Teacher defaultTeacher("IvanIvanovich", "Doctor");

Subject defaultSubject("Math");

std::shared\_ptr<Teacher> defaultSharedTeacher = std::make\_shared<Teacher>(defaultTeacher);

std::shared\_ptr<Subject> defaultSharedSubject = std::make\_shared<Subject>(defaultSubject);

defaultSharedSubject->setTeacher(defaultSharedTeacher);

defaultSharedSubject->deleteTeacher(defaultSharedTeacher);

std::shared\_ptr<Subject> actual = defaultSharedTeacher->getSubject();

std::shared\_ptr<Subject> expected = nullptr;

ASSERT\_EQ(actual, expected);

}

int main(int argc, char \*\*argv)

{

testing::InitGoogleTest(&argc, argv);

return RUN\_ALL\_TESTS();

}

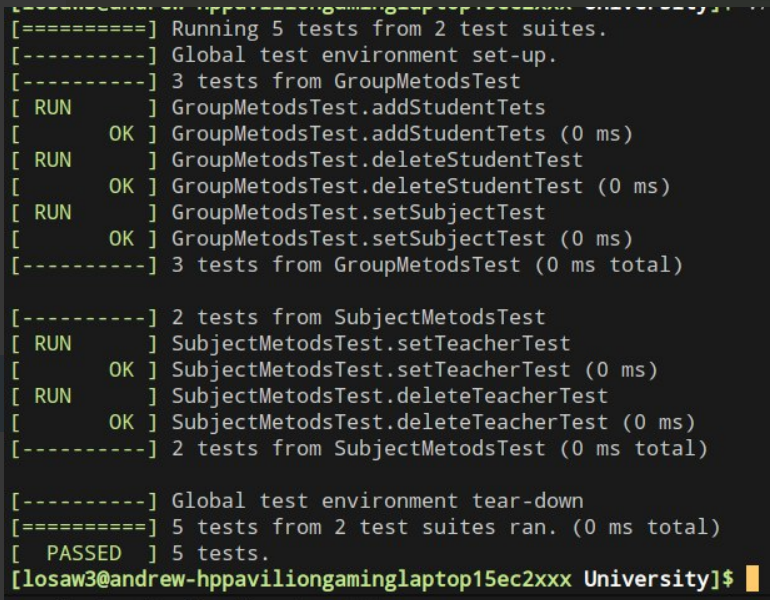


Рисунок 1 – прохождение тестов

**Uml- диаграмма**

Изображение выглядит как текст, снимок экрана, меню

Автоматически созданное описание

Рисунок 2 – Uml-диаграмма проекта



Рисунок 3 – approve задания