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

Институт транспортной техники и систем управления

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 ЛАБОРАТОРНОЙ РАБОТЕ №1

По дисциплине «ООП»

ВАРИАНТ 25

Выполнил: ст. гр. ТКИ-141

Блынский Герман Вадимович

Проверил:

Москва 2024

Оглавление

[1 Решение задачи 2](#_Toc168890148)

[1.2 UML диаграмма алгоритма 3](#_Toc168890149)

[1.3 Текст программы на языке С++ 4](#_Toc168890150)

[1.4 Результаты выполнения программы 14](#_Toc168890151)

[1.5 Выполнение тестовых примеров 15](#_Toc168890152)

[1.6 Отметка о выполнении задания в веб-хостинге системы контроля версий 18](#_Toc168890153)

1. Решение задачи

**1.1 Формулировка** **задачи**

Реализовать класс **Преподаватель**, дополнив члены базового класса полями должность (класс **Должность**), научное звание (класс **Научное звание**), научная степень (класс **Научная степень**). Определить статический метод чтения преподавателя из стандартного потока ввода и метод сериализации объекта в строку ToString(). Определить метод сравнения объекта **Преподаватель** по выбранному критерию (фамилия, должность, научное звание, научная степень).

Класс **Должность** содержит множество должностей, переопределены оператор сдвига вправо и метод сериализации в строку ToString().

Класс **Научная степень** содержит множество научных степеней,ьпереопределены оператор сдвига вправо и метод сериализации в строку ToString().

Класс **Научное звание** переопределены оператор сдвига вправо и метод сериализации в строку ToString().

1.2 UML диаграмма алгоритма

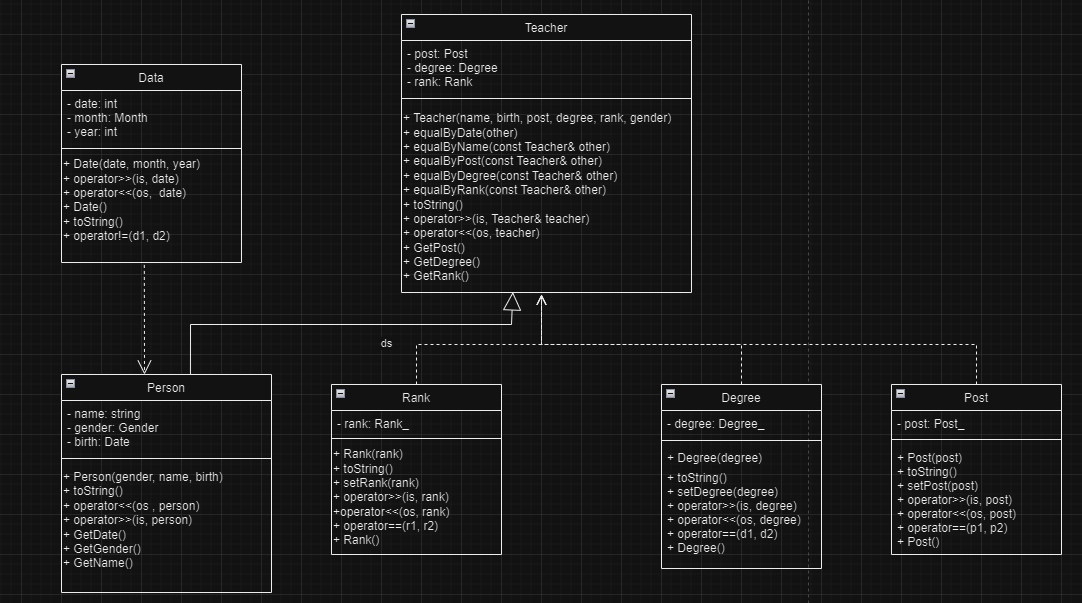
UML диаграмма представлена ниже (Рисунок 1).

Рисунок 1 - UML диаграмма 1

1.3 Текст программы на языке С++

//Файл Date.h

#pragma once

#include <iostream>

namespace uizi {

enum Month {January = 1,

February = 2, March = 3, April = 4, May = 5, June = 6, July = 7, August = 8, September = 9, October = 10, November = 11, December = 12};

class Date

{

/\*

\*@brief день рождения

\*/

int date;

/\*

\*@brief месяц рождения

\*/

Month month;

/\*

\*@brief год рождения

\*/

int year;

public:

/\*

\*@brief конструктор класса Date через ввод дня, месяца и года рождения

\*/

Date(int date, int month, int year);

/\*

\*@brief перегрузка оператора сдвига вправо

\*/

friend std::istream& operator>>(std::istream& is, Date& date) {

int dateInt = 0;

int monthInt = 0;

int yearInt = 0;

is >> dateInt >> monthInt >> yearInt;

if (dateInt <= 0 || dateInt > 31 || monthInt <= 0 || monthInt > 12 || yearInt < 0) {

throw std::invalid\_argument("wrong value date or month or year!");

}

date.date = dateInt;

date.month = (Month)monthInt;

date.year = yearInt;

return is;

}

/\*

\*@brief перегрузка оператора сдвига влево

\*/

friend std::ostream& operator<<(std::ostream& os, const Date& date) {

os << date.toString();

return os;

}

/\*

\*@brief Конструктор по умолчанию

\*/

Date();

/\*

\*@brief Метод класса для проебразования экземпляра класса Date в строку

\*/

std::string toString() const;

/\*

\*@brief перегрузка оператора равенства

\*/

friend bool operator==(const Date& d1, const Date& d2) {

return (d1.date == d2.date && d1.month == d2.month && d1.year == d2.year);

}

/\*

\*@brief перегрузка оператора неравенства

\*/

friend bool operator!=(const Date& d1, const Date& d2) {

return d1.date != d2.date || d1.month != d2.month || d1.year != d2.year;

}

};

}

//Файл Degree.h

#pragma once

#include <iostream>

namespace uizi {

enum Degree\_{DocentDegree = 0, ProfessorDegree = 1};

class Degree

{

/\*

\*@brief Научная степень

\*/

Degree\_ degree;

public:

/\*

\*@brief Конструктор класса Degree

\*/

Degree(enum Degree\_ degree);

/\*

\*@brief метод класса, преобразовающий экземпляр класса в строку

\*/

std::string toString() const;

/\*

\*@brief Сеттер класса Degree

\*/

void setDegree(Degree\_ degree);

/\*

\*@brief перегрузка оператора сдвига вправо

\*/

friend std::istream& operator>>(std::istream& is, Degree& degree) {

std::string degreeStr;

is >> degreeStr;

if (degreeStr == "Docent") {

degree.setDegree(DocentDegree);

return is;

}

if (degreeStr == "Professor") {

degree.setDegree(ProfessorDegree);

return is;

}

throw std::invalid\_argument("wrong value degree!");

}

/\*

\*@brief перегрузка оператора сдвига влево

\*/

friend std::ostream& operator<<(std::ostream& os, Degree& degree){

os << degree.toString();

return os;

}

/\*

\*@brief перегрузка оператора равенства

\*/

friend bool operator==(const Degree& d1, const Degree& d2) {

return d1.degree == d2.degree;

}

/\*

\*@brief Конструктор по умолчанию

\*/

Degree();

};

}

//Файл Post.h

#pragma once

#include <iostream>

namespace uizi {

enum Post\_{GreatPost = 0, DocentPost = 1, ProfessorPost = 2};

class Post

{

Post\_ post;

public:

Post(enum Post\_ post);

std::string toString();

void setPost(enum Post\_ post);

friend std::istream& operator>>(std::istream& is, Post &post){

std::string postStr;

is >> postStr;

if (postStr == "Great") {

post.setPost(GreatPost);

return is;

}

if (postStr == "Docent") {

post.setPost(DocentPost);

return is;

}

if (postStr == "Professor") {

post.setPost(ProfessorPost);

return is;

}

throw std::invalid\_argument("wrong value post!");

}

friend std::ostream& operator<<(std::ostream& os, Post &post){

os << post.toString();

return os;

}

friend bool operator==(const Post& p1, const Post& p2) {

return p1.post == p2.post;

}

Post();

};

}

//Файл Rank.h

#pragma once

#include <iostream>

namespace uizi {

enum Rank\_{NoRank = 0, CandidatRank = 1, DoctorRank = 2};

class Rank

{

Rank\_ rank;

public:

Rank(Rank\_ rank);

std::string toString() const;

void setRank(Rank\_ rank);

friend std::istream& operator>>(std::istream& is, Rank &rank) {

std::string rankStr;

is >> rankStr;

if (rankStr == "No rank") {

rank.setRank(NoRank);

return is;

}

if (rankStr == "Candidat") {

rank.setRank(CandidatRank);

return is;

}

if (rankStr == "Doctor") {

rank.setRank(DoctorRank);

return is;

}

}

friend std::ostream& operator<<(std::ostream& os, const Rank& rank) {

os << rank.toString();

return os;

}

friend bool operator==(const Rank& r1, const Rank& r2) {

return r1.rank == r2.rank;

}

Rank();

};

}

//Файл Person.h

#pragma once

#include <iostream>

#include "Date.h"

namespace uizi {

using FullName = std::string;

/\*

\*@brief Гендеры

\*/

enum Gender { Male = 0, Female = 1, NonBinary = 3 };

class Person

{

/\*

\*@brief Имя

\*/

FullName name;

/\*

\*@brief Гендеры

\*/

Gender gender;

/\*

\*@brief Дата рождения

\*/

Date birth;

public:

/\*

\*@brief Конструктор класса Person через ввод гендера, имя и даты рождения

\*/

Person(const Gender& gender, const FullName& name, const Date& birth);

/\*

\*@brief метод класса, преобразовывающий в строку

\*/

std::string toString() const;

/\*

\*@brief Перегрузка оператора сдвига влево

\*/

friend std::ostream& operator<<(std::ostream& os , const Person& person) {

os << person.toString();

return os;

}

/\*

\*@brief Перегрузка оператора сдвига вправо

\*/

friend std::istream& operator>>(std::istream& is, Person& person)

{

FullName name;

int gender;

Date birth;

if (is >> name >> gender >> birth) {

person = Person(static\_cast<Gender>(gender), name, birth);

}

return is;

}

Date GetDate() const;

Gender GetGender() const;

FullName GetName() const;

};

}

//Файл Teacher.h

#pragma once

#include "Person.h"

#include "Post.h"

#include "Degree.h"

#include "Rank.h"

namespace uizi {

class Teacher :

public Person

{

Post post;

Degree degree;

Rank rank;

public:

Teacher(const FullName& name, const Date& birth, const Post& post, const Degree& degree, const Rank& rank, const Gender& gender);

bool equalByDate(const Teacher& other) const;

bool equalByName(const Teacher& other) const;

bool equalByPost(const Teacher& other) const;

bool equalByDegree(const Teacher& other) const;

bool equalByRank(const Teacher& other) const;

std::string toString();

friend std::istream& operator>>(std::istream& is, Teacher& teacher) {

FullName name;

Date birth;

Post post;

Degree degree;

Rank rank;

int gender;

if (is >> name >> birth >> post >> degree >> rank >> gender) {

teacher = Teacher(name, birth, post, degree, rank, static\_cast<Gender>(gender));

}

}

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

os << teacher.toString();

return os;

}

Post GetPost() const;

Degree GetDegree() const;

Rank GetRank() const;

};

}

//Файл Date.cpp

#include "Date.h"

#include <string>

#include <sstream>

namespace uizi {

Date::Date(int date, int month, int year) {

if (date <= 0 || date > 31 || month <= 0 || month > 12 || year < 0) {

throw std::invalid\_argument("wrong value date or month or year!");

}

this->date = date;

this->month = (enum Month)month;

this->year = year;

}

Date::Date() {

this->date = 1;

this->month = January;

this->year = 1;

}

std::string Date::toString() const {

std::stringstream a{};

a << date << "." << month << "." << year;

return a.str();

}

}

//Файл Degree.cpp

#include "Degree.h"

void uizi::Degree::setDegree(Degree\_ degree) {

this->degree = degree;

}

uizi::Degree::Degree(enum Degree\_ degree) {

this->degree = degree;

}

std::string uizi::Degree::toString() const {

if (degree == DocentDegree) {

return "Docent";

}

if (degree == ProfessorDegree) {

return "Professor";

}

throw std::invalid\_argument("wrong value degree!");

}

uizi::Degree::Degree() {

this->degree = DocentDegree;

}

//Файл Post.cpp

#include "Post.h"

std::string uizi::Post::toString() {

if (post == GreatPost) {

return "Great";

}

if (post == DocentPost) {

return "Docent";

}

if (post == ProfessorPost) {

return "Professor";

}

throw;

}

void uizi::Post::setPost(enum Post\_ post) {

this->post = post;

}

uizi::Post::Post(enum Post\_ post) {

this->post = post;

}

uizi::Post::Post() {

this->post = GreatPost;

}

//Файл Rank.cpp

#include "Rank.h"

uizi::Rank::Rank(Rank\_ rank) {

this->rank = rank;

}

void uizi::Rank::setRank(Rank\_ rank) {

this->rank = rank;

}

std::string uizi::Rank::toString() const {

if (rank == NoRank) {

return "No rank";

}

if (rank == CandidatRank) {

return "Candidat";

}

if (rank == DoctorRank) {

return "Doctor";

}

throw std::invalid\_argument("wrong value rank!");

}

uizi::Rank::Rank() {

this->rank = NoRank;

}

//Файл Person.cpp

#include "Person.h"

#include "sstream"

uizi::Person::Person(const Gender& gender, const FullName& name, const Date& birth){

this->gender = gender;

this->name = name;

this->birth = birth;

}

std::string uizi::Person::toString() const

{

std::stringstream a{};

if (gender == Male) {

a << name << " " << "Male" << " " << birth << " ";

}

if (gender == Female) {

a << name << " " << "Female" << " " << birth << " ";

}

if (gender == NonBinary) {

a << name << " " << "NonBinary" << " " << birth << " ";

}

return a.str();

}

namespace uizi {

Date Person::GetDate() const {

return birth;

}

FullName Person::GetName() const {

return name;

}

Gender Person::GetGender() const {

return gender;

}

}

//Файл Teacher.cpp

#include "Teacher.h"

#include "Person.h"

#include "Post.h"

#include "Degree.h"

#include "Rank.h"

#include <sstream>

namespace uizi {

Teacher::Teacher(const FullName& name, const Date& birth, const Post& post, const Degree& degree, const Rank& rank, const Gender& gender)

:Person(gender, name, birth)

{

this->post = post;

this->degree = degree;

this->rank = rank;

}

bool Teacher::equalByDate(const Teacher& other) const

{

return this->GetDate() == other.GetDate();

}

bool Teacher::equalByName(const Teacher& other) const

{

return this->GetName() == other.GetName();

}

bool Teacher::equalByPost(const Teacher& other) const

{

return this->GetPost() == other.GetPost();

}

bool Teacher::equalByDegree(const Teacher& other) const

{

return this->GetDegree() == other.GetDegree();

}

bool Teacher::equalByRank(const Teacher& other) const

{

return this->GetRank() == other.GetRank();

}

Post Teacher::GetPost() const {

return post;

}

Degree Teacher::GetDegree() const {

return degree;

}

Rank Teacher::GetRank() const {

return rank;

}

}

std::string uizi::Teacher::toString()

{

std::stringstream a{};

a << Person::toString();

a << post << " " << degree << " " << rank;

return a.str();

}

//Файл main.cpp

#include <iostream>

#include <string>

#include "Teacher.h"

/\*\*

\* @brief Точка входа в программу

\* @return 0 при успешном выполнении

\*/

int main()

{

setlocale(LC\_ALL, "ru\_RU.UTF-8");

using namespace uizi;

std::string name = "Вася Пупкин";

Date birth = Date(23, 12, 1952);

Post post = Post(DocentPost);

Degree degree = Degree(DocentDegree);

Rank rank = Rank(CandidatRank);

Gender gender = Gender(Male);

Teacher teacher = Teacher(name, birth, post, degree, rank, gender);

std::string c = teacher.toString();

std::cout << c;

return 0;

}

1.4 Результаты выполнения программы

Результаты выполнения программы в C++ представлена ниже (Рисунок 2).

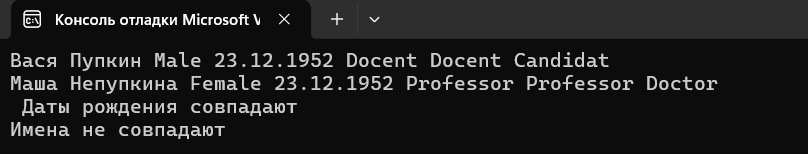


Рисунок 2 - Результаты выполнения программы

1.5 Выполнение тестовых примеров

Выполнены тестовые примеры. Результаты их выполнения представлены ниже (Рисунок 3–8).

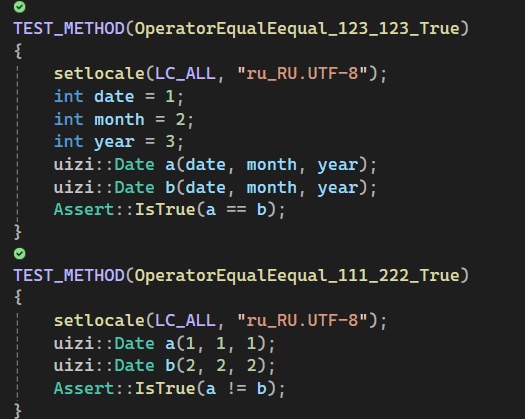


Рисунок 3 – Результаты работы тестов

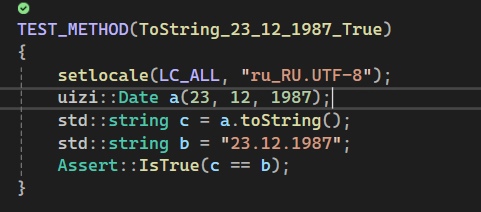


Рисунок 4 - Результаты работы тестов

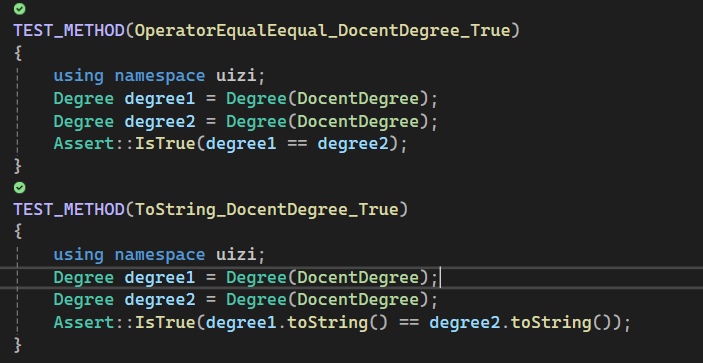


Рисунок 5 - Результаты работы тестов



Рисунок 6 – Результаты работы тестов



Рисунок 7 - Результаты выполнения тестов

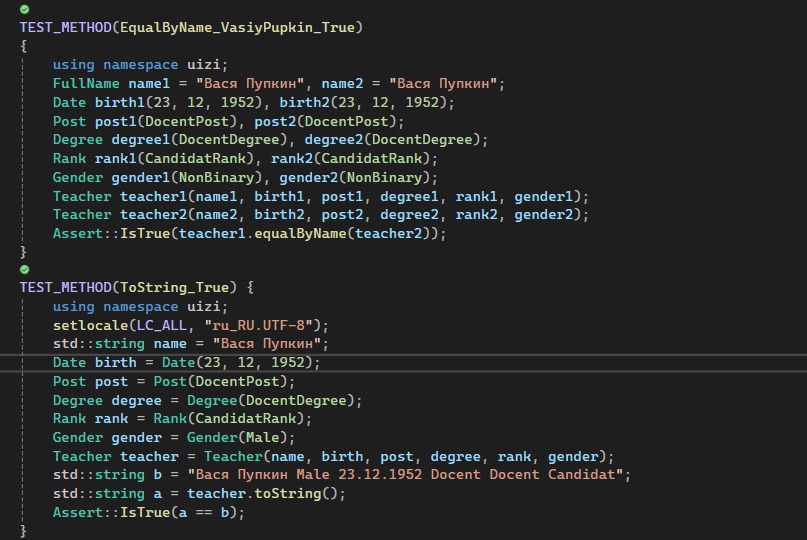


Рисунок 8 - Результаты выполнения тестов

1.6 Отметка о выполнении задания в веб-хостинге системы контроля версий

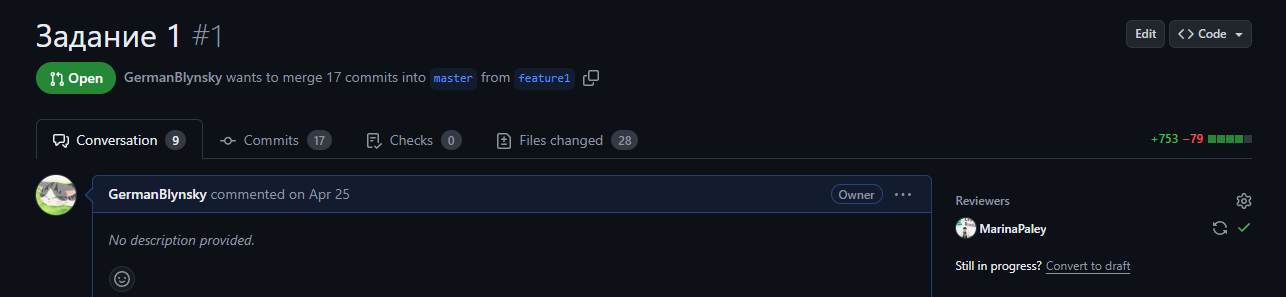


Рисунок 9 – Approve задачи