**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ  
РОССИЙСКОЙ ФЕДЕРАЦИИ**ФЕДЕРАЛЬНОЕ ГОСУДРАСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ОБРАЗОВАНИЯ  
**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ   
ТЕХНОЛОГИЧСЕКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»  
(БГТУ им В.Г.Шухова)**

Кафедра программного обеспечения вычислительной техники и   
автоматизированных систем

РГЗ  
дисциплина: **Объектно-ориентированное программирование**  
тема: «**Генеалогическое древо»**

Выполнил: ст. группы ПВ-22  
Беляев А. С.  
проверил: ст. пр. Буханов Д. Г.

Белгород 2020

**Задание**

1. Выполнить построение объектной модели предметной области.

2. Разработать диаграмму классов предметной области предложенного варианта.

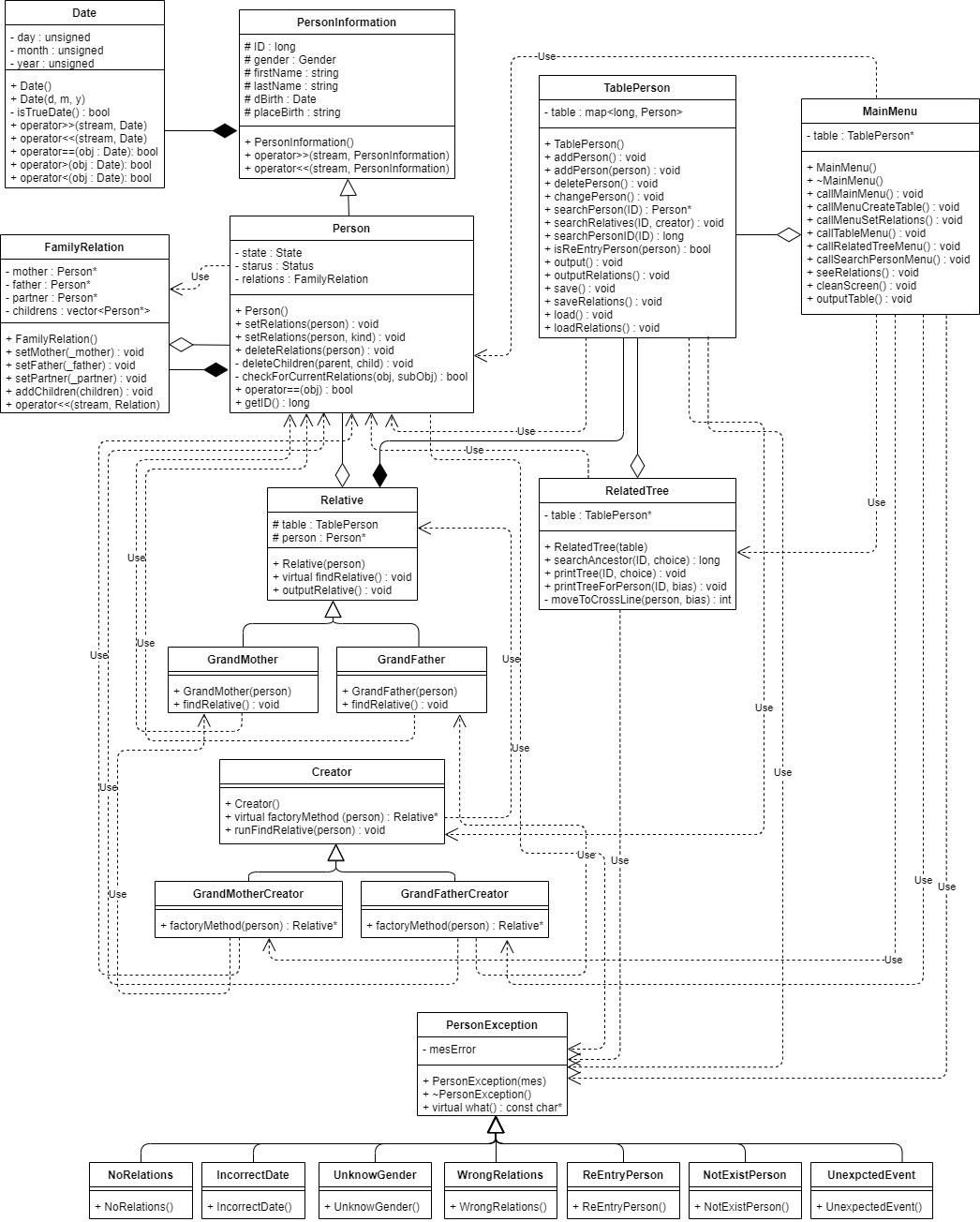
3. Выполнить реализацию описанного задания.

4. Оформить отчет.

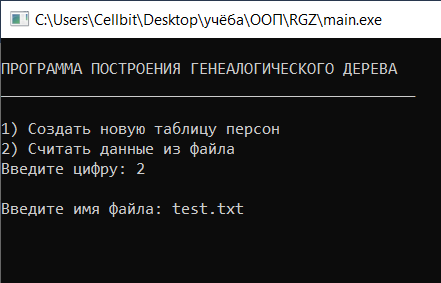
**Задание варианта №7**

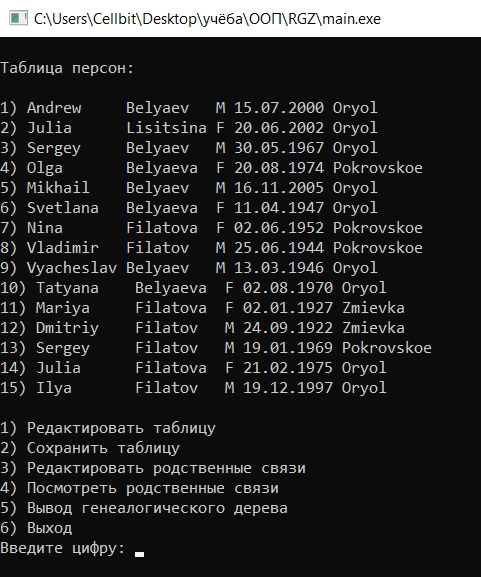
Программа для построения генеалогического дерева. Программа должна обеспечивать возможность ввода информации о людях, состоящих в родстве друг с другом (пол, дата рождения, место рождения, основной вид деятельности и т.п.). Для каждого человека должна быть возможность указания его родителей, детей. Для регистрации изменений в программе должны быть предусмотрены события различного типа: рождение, смерть, женитьба и т.п. По данным о родственных связях программа должна обеспечить вывода на экран генеалогического дерева.

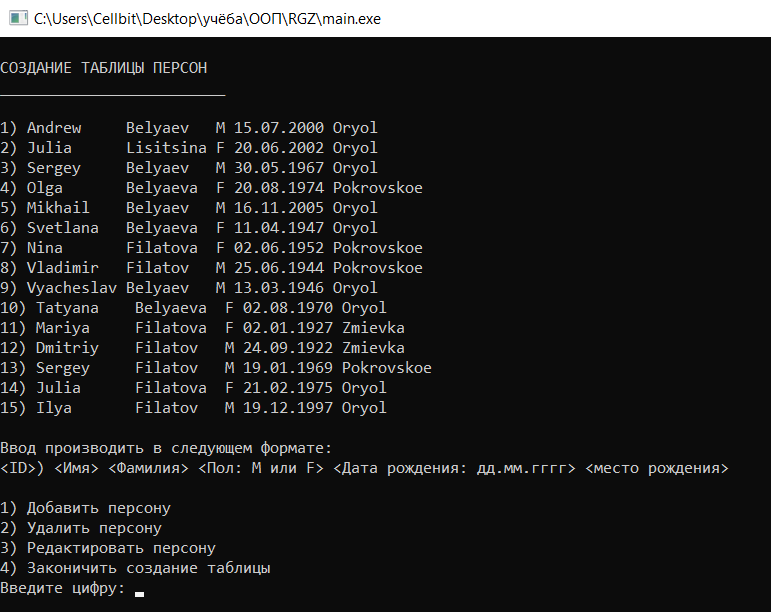
**Диаграмма классов**

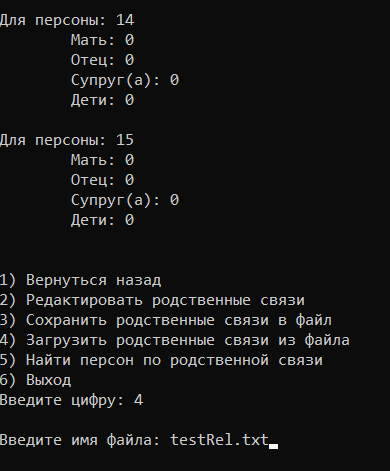
****

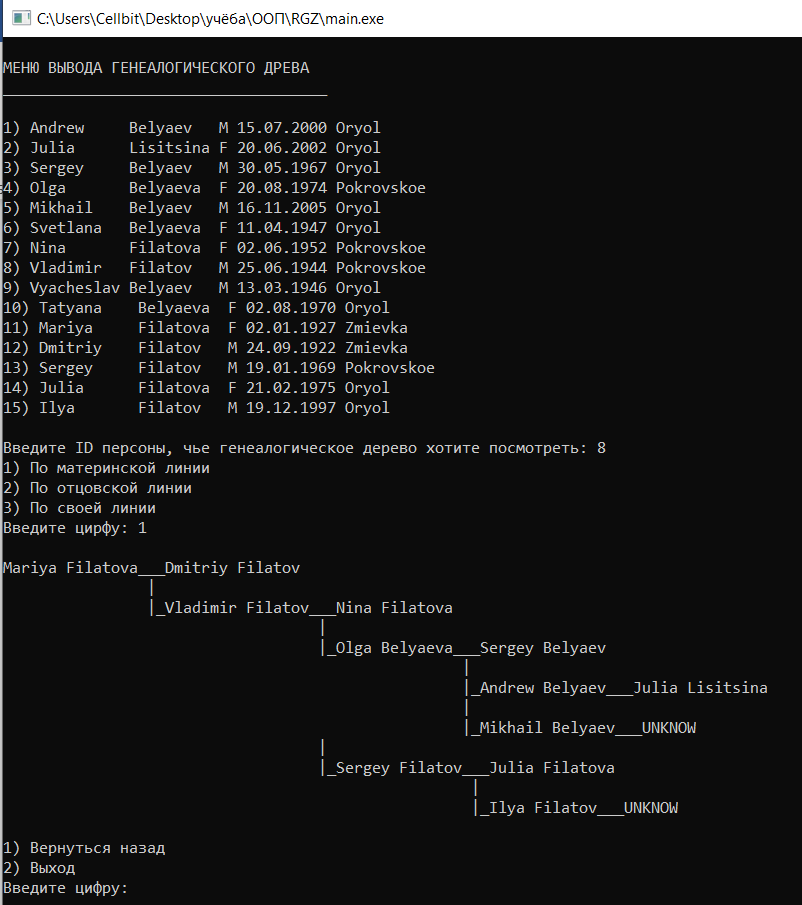
**Скриншоты выполнения программы**

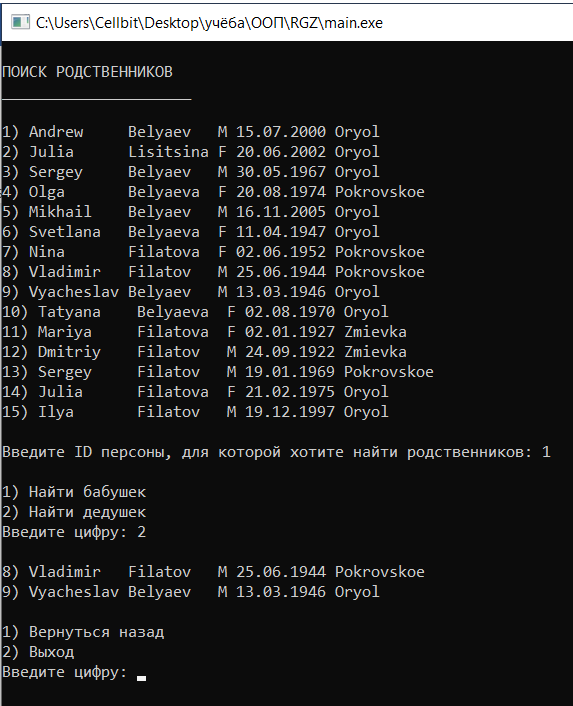
****

****









**Код программы**

*Содержание файла person.hpp:*

#ifndef PERSON\_H

#define PERSON\_H

#include <iostream>

#include <string>

#include <vector>

#include <iterator>

#include <iomanip>

#include "personException.hpp"

using namespace std;

enum Gender { MALE, FEMALE };

enum State { ALIVE, DEAD, UNKNOW };

enum Status { MARRIED, NOT\_MARRIED };

enum ChoiceSetRelations { SET\_MOTHER = 1, SET\_FATHER, SET\_PARTNER, ADD\_CHILDREN };

const unsigned CURRENT\_YEAR = 2020;

class Date {

    unsigned day;

    unsigned month;

    unsigned year;

    bool isTrueDate();

public:

    Date() {};

    Date(unsigned d, unsigned m, unsigned y) : day(d), month(m), year(y) {};

    friend istream& operator>>(istream& stream, Date& date);

    friend ostream& operator<<(ostream& stream, Date& date);

    bool operator==(Date& obj);

    bool operator>(Date& obj);

    bool operator<(Date& obj);

};

class PersonInformation {

protected:

    long ID;

    Gender gender;

    string firstName;

    string lastName;

    Date dBirth;

    string placeBirth;

public:

    PersonInformation() {};

    friend istream& operator>>(istream& stream, PersonInformation& person);

    friend ostream& operator<<(ostream& stream, PersonInformation& person);

};

class Person;

class FamilyRelation {

    Person\* mother;

    Person\* father;

    Person\* partner;

    vector<Person\*> childrens;

public:

    FamilyRelation() : mother(nullptr), father(nullptr), partner(nullptr) {};

    void setMother(Person\* \_mother) { mother = \_mother; };

    void setFather(Person\* \_father) { father = \_father; };

    void setPartner(Person\* \_partner) { partner = \_partner;};

    void addChildren(Person\* children) { childrens.push\_back(children); };

    friend ostream& operator<<(ostream& stream, FamilyRelation& relation);

    friend class Person;

    friend class RelatedTree;

    friend class GrandMother;

    friend class GrandFather;

};

class Person : public PersonInformation {

    State state;

    Status status;

    FamilyRelation relations;

    void deleteChildren(Person\* parent, Person\* child);

    bool checkForCorrectRelations(Person\* obj, Person\* subObj, short choice);

public:

    Person() : PersonInformation(), state(UNKNOW), status(NOT\_MARRIED) {};

    void setRelations(Person\* person);

    void setRelations(Person\* person, short kindOfSet);

    void deleteRelations(Person\* person);

    bool operator==(Person& obj);

    long getID() { return ID; };

    friend class TablePerson;

    friend class RelatedTree;

    friend class GrandMother;

    friend class GrandFather;

};

#endif

*Содержание файла person.cpp:*

#include "person.hpp"

bool Date::isTrueDate() {

    return ((day > 0 && day <= 31) &&

            (month > 0 && month <= 12) &&

            (year > 0 && year <= CURRENT\_YEAR));

}

istream& operator>>(istream& stream, Date& date) {

    stream >> date.day;

    stream.ignore();

    stream >> date.month;

    stream.ignore();

    stream >> date.year;

    if (!date.isTrueDate())

        throw IncorrectDate();

    return stream;

}

ostream& operator<<(ostream& stream, Date& date) {

    stream << setw(2) << right << setfill('0') << date.day << '.'

           << setw(2) << right << setfill('0') << date.month << '.'

           << setw(4) << right << setfill('0') << date.year;

    return stream;

}

bool Date::operator==(Date& obj) {

    return (this->day == obj.day && this->month == obj.month &&

this->year == obj.year);

}

bool Date::operator>(Date& obj) {

    return (this->year > obj.year);

}

bool Date::operator<(Date& obj) {

    return (this->year < obj.year);

}

istream& operator>>(istream& stream, PersonInformation& person) {

    stream >> person.ID;

    stream.ignore();

    stream >> person.firstName >> person.lastName;

    char gender;

    string str;

    stream >> gender;

    if (toupper(gender) == 'M')

        person.gender = MALE;

    else if (toupper(gender) == 'F')

        person.gender = FEMALE;

    else

        throw UnknowGender();

    stream >> person.dBirth;

    stream >> person.placeBirth;

    return stream;

}

ostream& operator<<(ostream& stream, PersonInformation& person) {

    stream << person.ID << ") ";

    stream << setw(10) << left << setfill(' ') << person.firstName << ' '

           << setw(10) << left << setfill(' ') << person.lastName;

    if (person.gender == MALE)

        stream << "M ";

    else if (person.gender == FEMALE)

        stream << "F ";

    stream << person.dBirth << ' ';

    stream << person.placeBirth;

    return stream;

}

ostream& operator<<(ostream& stream, FamilyRelation& relations) {

    stream << "\n\tМать: ";

    if (relations.mother != nullptr)

        stream << relations.mother->getID();

    else

        stream << 0;

    stream << "\n\tОтец: ";

    if (relations.father != nullptr)

        stream << relations.father->getID();

    else

        stream << 0;

    stream << "\n\tСупруг(а): ";

    if (relations.partner != nullptr)

        stream << relations.partner->getID();

    else

        stream << 0;

    stream << "\n\tДети: ";

    if (relations.childrens.size() != 0) {

        for (auto children : relations.childrens)

            stream << children->getID() << ' ';

    }

    else

        stream << 0;

    return stream;

}

bool Person::operator==(Person& obj) {

    return (this->firstName == obj.firstName && this->lastName == obj.lastName &&

            this->dBirth == obj.dBirth && this->placeBirth == obj.placeBirth);

}

void Person::setRelations(Person\* person) {

    cout << "\nВыберете вид связи:\n";

    cout << "\n1) Мать\n2) Отец\

             \n3) Супруг\n4) Ребёнок\

             \nВведите цифру: ";

    short kindOfSet;

    cin >> kindOfSet;

    if (checkForCorrectRelations(this, person, kindOfSet)) {

        switch (kindOfSet)

        {

        case SET\_MOTHER: {

            relations.setMother(person);

            person->relations.addChildren(this);

            break;

        }

        case SET\_FATHER: {

            relations.setFather(person);

            person->relations.addChildren(this);

            break;

        }

        case SET\_PARTNER: {

            relations.setPartner(person);

            person->relations.setPartner(this);

            person->status = MARRIED;

            this->status = MARRIED;

            break;

        }

        case ADD\_CHILDREN: {

            relations.addChildren(person);

            if (this->gender == MALE)

                person->relations.setFather(this);

            else if (this->gender == FEMALE)

                person->relations.setMother(this);

            break;

        }

        default:

            throw UnexpectedEvent();

        }

    }

    else

        throw WrongRelations();

}

void Person::setRelations(Person\* person, short kindOfSet) {

    if (checkForCorrectRelations(this, person, kindOfSet)) {

        switch (kindOfSet)

        {

        case SET\_MOTHER: {

            relations.setMother(person);

            person->relations.addChildren(this);

            break;

        }

        case SET\_FATHER: {

            relations.setFather(person);

            person->relations.addChildren(this);

            break;

        }

        case SET\_PARTNER: {

            relations.setPartner(person);

            person->relations.setPartner(this);

            person->status = MARRIED;

            this->status = MARRIED;

            break;

        }

        case ADD\_CHILDREN: {

            relations.addChildren(person);

            if (this->gender == MALE)

                person->relations.setFather(this);

            else if (this->gender == FEMALE)

                person->relations.setMother(this);

            break;

        }

        default:

            throw UnexpectedEvent();

        }

    }

}

bool Person::checkForCorrectRelations(Person\* obj, Person\* subObj, short choice) {

    //нельзя установить родственную связь с самим собой

    if (\*obj == \*subObj)

        return false;

    //нельзя установить персону больше чем на одну родственную связь

    if (obj->relations.mother == subObj || obj->relations.father == subObj ||

        obj->relations.partner == subObj)

            return false;

    for (auto children : obj->relations.childrens)

        if (\*children == \*subObj)

            return false;

    switch (choice)

    {

    case SET\_MOTHER:

        //мать обязательно - женского пола и старше сына, а ребенок не имеет матери

        return (subObj->gender == FEMALE && obj->dBirth > subObj->dBirth &&

                obj->relations.mother == nullptr);

    case SET\_FATHER:

        //отец обязательно мужского пола и старше сына, а ребенок не имеет отца

        return (subObj->gender == MALE && obj->dBirth > subObj->dBirth &&

                obj->relations.father == nullptr);

    case SET\_PARTNER:

        //супруги обязательно разных полов, и оба не состоят в браке

        return !(subObj->gender == obj->gender) && obj->relations.partner == nullptr &&

                subObj->relations.partner == nullptr;

    case ADD\_CHILDREN:

        //сын обязательно младше родителя

        if (obj->dBirth < subObj->dBirth) {

            if (obj->gender == MALE) //и если это отец, то ребенок не имеет сына

                return (subObj->relations.father == nullptr);

            else if (obj->gender == FEMALE)

//и если это мать, то ребенок не имеет матери

                return (subObj->relations.mother == nullptr);

        }

        else

            return false;

    default:

        throw UnexpectedEvent();

    }

}

void Person::deleteChildren(Person\* parent, Person\* child) {

    vector<Person\*>::iterator children = parent->relations.childrens.begin();

    while (children != parent->relations.childrens.end()) {

        if (\*children == child){

            parent->relations.childrens.erase(children);

            break;

        }

        children++;

    }

}

void Person::deleteRelations(Person\* person) {

    if (this->relations.mother == person) {

        this->relations.mother = nullptr;

        deleteChildren(person, this);

    }

    else if (this->relations.father == person) {

        this->relations.father = nullptr;

        deleteChildren(person, this);

    }

    else if (this->relations.partner == person) {

        this->relations.partner = nullptr;

        person->relations.partner = nullptr;

        person->status = NOT\_MARRIED;

        this->status = NOT\_MARRIED;

    }

    else if (person->relations.mother == this ||

person->relations.father == this) {

        deleteChildren(this, person);

        if (this->gender == MALE)

            person->relations.father = nullptr;

        else if (this->gender == FEMALE)

            person->relations.mother = nullptr;

    }

    else

        throw NoRelations();

}

*Содержание файла tablePerson.hpp:*

#ifndef TABLE\_H

#define TABLE\_H 1

#include <iostream>

#include <map>

#include <fstream>

#include "person.hpp"

using namespace std;

class Creator;

class TablePerson {

    map<long, Person> table;

    void setRelationFromFile(ifstream& file, long setPersonID, Person\* setPerson,  Person\* currentPerson, short choice);

public:

    TablePerson() {};

    void addPerson();

    void addPerson(Person person);

    void deletePerson();

    void changePerson();

    Person\* searchPerson(long ID);

    void searchRelatives(long personID, Creator\* creator);

    long searchPersonID(long ID);

    bool isReEntryPerson(Person person);

    void output();

    void outputRelations();

    void save();

    void saveRelations();

    void load();

    void loadRelations();

};

#endif

*Содержание файла tablePerson.cpp:*

#include "tablePerson.hpp"

#include "relatedTree.hpp"

#include "mainMenu.hpp"

void TablePerson::addPerson() {

    Person newPerson;

    cout << "\nВведите информацию о персоне:\n";

    cin >> newPerson;

    if (isReEntryPerson(newPerson))

        throw ReEntryPerson();

    table[newPerson.getID()] = newPerson;

}

void TablePerson::addPerson(Person person) {

    if (isReEntryPerson(person))

        throw ReEntryPerson();

    table[person.getID()] = person;

}

void TablePerson::deletePerson() {

    long ID;

    cout << "\nВведите ID персоны, которую хотите удалить: ";

    cin >> ID;

    if (table.count(ID) != 0) {

        table.erase(ID);

        cout << "\nПерсона удалена!\n";

    }

    else

        throw NotExistPerson();

}

void TablePerson::changePerson() {

    long ID;

    cout << "\nВведите ID персоны, которую хотите отредактировать: ";

    cin >> ID;

    if (table.count(ID) != 0) {

        cout << "\n1) Изменить имя\n2) Изменить фамилию\n3) Изменить пол\n4) Изменить дату рождения\n5) Изменить место рождения\nВведите цифру: ";

        short choice;

        cin >> choice;

        Person\* person = searchPerson(ID);

        switch (choice)

        {

        case FIRST: {

            cout << "\nВведите новое имя: ";

            cin >> person->firstName;

            break;

        }

        case SECOND: {

            cout << "\nВведите новую фамилию: ";

            cin >> person->lastName;

            break;

        }

        case THIRD: {

            if (person->gender == MALE)

                person->gender = FEMALE;

            else

                person->gender = MALE;

            break;

        }

        case FOURTH: {

            cout << "\nВведите новую дату рождения: ";

            cin >> person->dBirth;

            break;

        }

        case FIFTH: {

            cout << "\nВведите новое место рождения: ";

            cin >> person->placeBirth;

            break;

        }

        default:

            UnexpectedEvent();

        }

        cout << "\nПерсона изменена!\n";

    }

    else

        throw NotExistPerson();

}

Person\* TablePerson::searchPerson(long ID) {

    if (table.count(ID) != 0)

        return &table[ID];

    else

        return nullptr;

}

void TablePerson::searchRelatives(long personID, Creator\* creator) {

    Person\* person = searchPerson(personID);

    if (person != nullptr)

        creator->runFindRelative(person);

    else

        NotExistPerson();

}

long TablePerson::searchPersonID(long ID) {

    if (table.count(ID) != 0)

        return table[ID].getID();

    else

        return 0;

}

bool TablePerson::isReEntryPerson(Person person) {

    for (auto sPerson : table)

        if (sPerson.second == person)

            return true;

    return false;

}

void TablePerson::output() {

    cout << endl;

    for (auto person : table)

        cout << person.second << endl;

}

void TablePerson::save() {

    string fileName;

    cout << "\nВведите имя файла: ";

    cin >> fileName;

    ofstream file(fileName);

    for (auto person : table)

        file << endl << person.second;

    file.close();

}

void TablePerson::load() {

    string fileName;

    cout << "\nВведите имя файла: ";

    cin >> fileName;

    ifstream file(fileName);

    Person newPerson;

    while (!file.eof()) {

        file >> newPerson;

        if (isReEntryPerson(newPerson)) {

            throw ReEntryPerson();

        }

        table[newPerson.getID()] = newPerson;

    }

    file.close();

}

void TablePerson::outputRelations() {

    for (auto person : table) {

        cout << "Для персоны: " << person.second.getID();

        cout << person.second.relations << endl << endl;

    }

}

void TablePerson::saveRelations() {

    string fileName;

    cout << "\nВведите имя файла: ";

    cin >> fileName;

    ofstream file(fileName);

    for (auto person : table) {

        file << person.second.getID();

        file << person.second.relations << endl << endl;

    }

    file.close();

}

void TablePerson::setRelationFromFile(ifstream& file, long setPersonID, Person\* setPerson, Person\* currentPerson, short choice) {

    string temp;

    file >> temp >> setPersonID;

    setPerson = searchPerson(setPersonID);

    if (setPerson != nullptr)

        currentPerson->setRelations(setPerson, choice);

}

void TablePerson::loadRelations() {

    string fileName;

    cout << "\nВведите имя файла: ";

    cin >> fileName;

    ifstream file(fileName);

    while (!file.eof()) {

        long currentPersonID, setPersonID;

        Person \*currentPerson, \*setPerson;

        file >> currentPersonID;

        currentPerson = searchPerson(currentPersonID);

        setRelationFromFile(file, setPersonID, setPerson, currentPerson,

SET\_MOTHER);

        setRelationFromFile(file, setPersonID, setPerson, currentPerson,

SET\_FATHER);

        setRelationFromFile(file, setPersonID, setPerson, currentPerson,

SET\_PARTNER);

        setRelationFromFile(file, setPersonID, setPerson, currentPerson,

ADD\_CHILDREN);

        char c;

        file.get(c);

        while (c != '\n') {

            setPerson = searchPerson(setPersonID);

            if (setPerson != nullptr)

                currentPerson->setRelations(setPerson, ADD\_CHILDREN);

            file.get(c);

        }

    }

    file.close();

}

*Содержание файла relatedTree.hpp:*

#ifndef RTREE\_H

#define RTREE\_H 1

#include <iostream>

#include "tablePerson.hpp"

using namespace std;

enum sideOfTree { ON\_MOTHER\_SIDE = 1, ON\_FATHER\_SIDE, ON\_OWN\_SIDE };

class RelatedTree {

    TablePerson\* table;

    int moveToCrossLine(Person\* person, int bias);

public:

    RelatedTree(TablePerson\* \_table) : table(\_table) {};

    long searchAncestor(long personID, short choice);

    void printTree(long personID, short choice);

    void printTreeForPerson(long personID, int bias);

};

class Relative {

protected:

    TablePerson table;

    Person\* person;

public:

    Relative(Person\* \_person) : person(\_person) {};

    virtual void findRelative() = 0;

    void outputRelative() { table.output(); };

};

class GrandMother : public Relative {

public:

    GrandMother(Person\* \_person) : Relative(\_person) {};

    void findRelative();

};

class GrandFather : public Relative {

public:

    GrandFather(Person\* \_person) : Relative(\_person) {};

    void findRelative();

};

class Creator {

public:

    Creator() {};

    virtual Relative\* factoryMethod(Person\* person) = 0;

    void runFindRelative(Person\* person);

};

class GrandMotherCreator : public Creator {

public:

    Relative\* factoryMethod(Person\* person) { return new GrandMother(person); };

};

class GrandFatherCreator : public Creator {

public:

    Relative\* factoryMethod(Person\* person) { return new GrandFather(person); };

};

#endif

*Содержание файла relatedTree.cpp:*

#include "relatedTree.hpp"

long RelatedTree::searchAncestor(long personID, short choice) {

    Person\* person = table->searchPerson(personID);

    if (person != nullptr) {

        if (choice == ON\_MOTHER\_SIDE)

            if (person->relations.mother != nullptr)

                return person->relations.mother->getID();

            else

                return 0;

        else if (choice == ON\_FATHER\_SIDE)

            if (person->relations.father != nullptr)

                return person->relations.father->getID();

            else

                return 0;

    }

    else

        throw NotExistPerson();

}

int RelatedTree::moveToCrossLine(Person\* person, int bias) {

    int lengthOfName = person->firstName.length() +

person->lastName.length() + 1;

    for (int i = 0; i < lengthOfName + bias; i++)

        cout << ' ';

    return lengthOfName;

}

void RelatedTree::printTreeForPerson(long personID, int bias) {

    Person\* person = table->searchPerson(personID);

    if (person != nullptr) {

        cout << person->firstName << ' ' << person->lastName;

        if (person->relations.partner != nullptr) {

            cout << "\_\_\_" << person->relations.partner->firstName << ' '

                << person->relations.partner->lastName << endl;

        }

        else

            cout << "\_\_\_UNKNOW\n";

        if (person->relations.childrens.size() != 0) {

            int curBias;

            for (auto children : person->relations.childrens) {

                moveToCrossLine(person, bias);

                cout << " |\n";

                curBias = moveToCrossLine(person, bias);

                cout << " |\_";

                long childrenID = children->getID();

                printTreeForPerson(childrenID, bias + curBias + 3);

            }

        }

    }

}

void RelatedTree::printTree(long personID, short choice) {

    if (choice == ON\_FATHER\_SIDE || choice == ON\_MOTHER\_SIDE) {

        long ancestorID = searchAncestor(personID, choice);

        if (ancestorID != 0) {

            cout << endl;

            printTreeForPerson(ancestorID, 0);

        }

        else

            throw NotExistPerson();

    }

    else if (choice == ON\_OWN\_SIDE) {

        if (table->searchPersonID(personID) != 0) {

            cout << endl;

            printTreeForPerson(personID, 0);

        }

        else

            throw NotExistPerson();

    }

}

void GrandMother::findRelative() {

    if (person->relations.mother->relations.mother != nullptr)

        table.addPerson(\*person->relations.mother->relations.mother);

    if (person->relations.father->relations.mother != nullptr)

        table.addPerson(\*person->relations.father->relations.mother);

}

void GrandFather::findRelative() {

    if (person->relations.mother->relations.father != nullptr)

        table.addPerson(\*person->relations.mother->relations.father);

    if (person->relations.father->relations.father != nullptr)

        table.addPerson(\*person->relations.father->relations.father);

}

void Creator::runFindRelative(Person\* person) {

    Relative\* relative = this->factoryMethod(person);

    relative->findRelative();

    relative->outputRelative();

    delete relative;

}

*Содержание файла meinMenu.hpp:*

#ifndef MENU\_H

#define MENU\_H

#include <iostream>

#include <limits>

#include "relatedTree.hpp"

using namespace std;

enum Actions {FIRST = 1, SECOND, THIRD, FOURTH, FIFTH, SIXTH };

class MainMenu {

    TablePerson\* table;

public:

    MainMenu() { table = new TablePerson; };

    ~MainMenu() { delete table; };

    void callMainMenu();

    void callMenuCreateTable();

    void callMenuSetRelations();

    void callTableMenu();

    void callRelatedTreeMenu();

    void callSearchPersonMenu();

    void seeRelations();

    void cleanScreen() { system("cls"); };

    void outputTable() { table->output(); };

};

#endif

*Содержание файла mainMenu.cpp:*

#include "mainMenu.hpp"

void MainMenu::callMainMenu() {

    cleanScreen();

    cout << "\nПРОГРАММА ПОСТРОЕНИЯ ГЕНЕАЛОГИЧЕСКОГО ДЕРЕВА\n";

    cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

    cout << "\n1) Создать новую таблицу персон\n2) Считать данные из файла\n

Введите цифру: ";

    short choice;

    cin >> choice;

    switch (choice)

    {

    case FIRST: {

        callMenuCreateTable();

        break;

    }

    case SECOND: {

        table->load();

        callTableMenu();

        break;

    }

    default:

        throw UnexpectedEvent();

    }

}

void MainMenu::callTableMenu() {

    cleanScreen();

    cout << "\nТаблица персон:\n";

    table->output();

    cout <<"\n1) Редактировать таблицу\n2) Сохранить таблицу\n3) Редактировать

родственные связи\n4) Посмотреть родственные связи\n5) Вывод

генеалогического дерева\n6) Выход\nВведите цифру: ";

    short choice;

    cin >> choice;

    switch (choice)

    {

    case FIRST: {

        callMenuCreateTable();

        break;

    }

    case SECOND: {

        table->save();

        cout << "\nТаблица сохранена!\n";

        cout << "\n1) Продолжить работу с таблицей\n2) Выход\nВведите цифру: ";

        cin >> choice;

        if (choice == FIRST)

            callTableMenu();

        else

            exit(0);

        break;

    }

    case THIRD: {

        callMenuSetRelations();

        break;

    }

    case FOURTH: {

        seeRelations();

        break;

    }

    case FIFTH: {

        callRelatedTreeMenu();

        break;

    }

    case SIXTH: {

        exit(0);

    }

    default:

        throw UnexpectedEvent();

    }

}

void MainMenu::callMenuCreateTable() {

    cleanScreen();

    cout << "\nСОЗДАНИЕ ТАБЛИЦЫ ПЕРСОН\n";

    cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

    outputTable();

    cout << endl;

    cout << "Ввод производить в следующем формате: " << endl;

    cout << "<ID>) <Имя> <Фамилия> <Пол: M или F> <Дата рождения: дд.мм.гггг>

<место рождения>\n";

    short choice;

    do {

        try {

            cout << "\n1) Добавить персону\n2) Удалить персону\n3) Редактировать персону\n4) Законичить создание таблицы\nВведите цифру: ";

            cin >> choice;

            if (choice == FIRST)

                table->addPerson();

            else if(choice == SECOND)

                table->deletePerson();

            else if (choice == THIRD)

                table->changePerson();

        }

        catch (PersonException exception) {

            cout << endl << exception.what() << endl;

            cin.ignore(numeric\_limits<streamsize>::max(), '\n');

        }

    } while (choice != FOURTH);

    callTableMenu();

}

void MainMenu::callMenuSetRelations() {

    cleanScreen();

    cout << "\nРЕДАКТИРОВАНИЕ РОДСТВЕННЫХ СВЯЗЕЙ\n";

    cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

    outputTable();

    cout << endl;

    long currentPersonID;

    cout << "Введите ID персоны для которой хотите настроить родственные

отношения: ";

    cin >> currentPersonID;

    Person\* currentPerson = table->searchPerson(currentPersonID);

    long setPersonID;

    cout << "Введите ID персоны, которая связана с персоной выше: ";

    cin >> setPersonID;

    Person\* setPerson = table->searchPerson(setPersonID);

    cout << "\n1) Установить родственные связи\n2) Удалить родственные связи\n

Введите цифру: ";

    short choice;

    cin >> choice;

    try {

        if (currentPerson == nullptr || setPerson == nullptr)

            throw NotExistPerson();

        switch (choice)

        {

        case FIRST: {

            currentPerson->setRelations(setPerson);

            break;

        }

        case SECOND: {

            currentPerson->deleteRelations(setPerson);

            break;

        }

        default:

            throw UnexpectedEvent();

        }

    }

    catch (PersonException exception) {

        cout << endl << exception.what() << endl;

        system("pause");

    }

    callTableMenu();

}

void MainMenu::seeRelations() {

    cleanScreen();

    cout << "\nРОДСТВЕННЫЕ СВЯЗИ\n";

    cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

    outputTable();

    cout << endl;

    table->outputRelations();

    cout << "\n1) Вернуться назад\n2) Редактировать родственные связи\n

3) Сохранить родственные связи в файл\n4) Загрузить родственные связи из

файла\n5) Найти персон по родственной связи\n6) Выход\nВведите цифру: ";

    short choice;

    cin >> choice;

    switch (choice)

    {

    case FIRST: {

        callTableMenu();

        break;

    }

    case SECOND: {

        callMenuSetRelations();

        break;

    }

    case THIRD: {

        table->saveRelations();

        cout << "\nТаблица сохранена!\n";

        callTableMenu();

        system("pause");

    }

    case FOURTH: {

        table->loadRelations();

        callTableMenu();

        break;

    }

    case FIFTH: {

        callSearchPersonMenu();

    }

    case SIXTH: {

        exit(0);

    }

    default:

        throw UnexpectedEvent();

    }

}

void MainMenu::callRelatedTreeMenu() {

    cleanScreen();

    cout << "\nМЕНЮ ВЫВОДА ГЕНЕАЛОГИЧЕСКОГО ДРЕВА\n";

    cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

    outputTable();

    cout << endl;

    cout << "Введите ID персоны, чье генеалогическое дерево хотите посмотреть: ";

    short personID;

    cin >> personID;

    cout << "1) По материнской линии\n2) По отцовской линии\n3) По своей линии\n

Введите цирфу: ";

    short choice;

    cin >> choice;

    try {

        RelatedTree tree(table);

        tree.printTree(personID, choice);

    }

    catch (PersonException exception) {

        cout << endl << exception.what() << endl;

    }

    cout << "\n1) Вернуться назад\n2) Выход\nВведите цифру: ";

    cin >> choice;

    if (choice == FIRST)

        callTableMenu();

    else if (choice == SECOND)

        exit(0);

};

void MainMenu::callSearchPersonMenu() {

    cleanScreen();

    cout << "\nПОИСК РОДСТВЕННИКОВ\n";

    cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

    outputTable();

    cout << endl;

    cout << "Введите ID персоны, для которой хотите найти родственников: ";

    short personID;

    cin >> personID;

    cout << "\n1) Найти бабушек\n2) Найти дедушек\nВведите цифру: ";

    short choice;

    cin >> choice;

    Creator\* creator;

    if (choice == FIRST) {

        creator = new GrandMotherCreator();

        table->searchRelatives(personID, creator);

    }

    else if (choice == SECOND) {

        creator = new GrandFatherCreator();

        table->searchRelatives(personID, creator);

    }

    delete creator;

    cout << "\n1) Вернуться назад\n2) Выход\nВведите цифру: ";

    cin >> choice;

    if (choice == FIRST)

        callTableMenu();

    else if (choice == SECOND)

        exit(0);

}

*Содержание файла personException.hpp:*

#ifndef EXCP\_H

#define EXCP\_H

#include <iostream>

#include <string>

using namespace std;

class PersonException : public exception {

    string mesError;

public:

    PersonException(string \_mes) : mesError(\_mes) {};

    ~PersonException() = default;

    virtual const char\* what() { return mesError.c\_str(); };

};

class IncorrectDate : public PersonException {

public:

    IncorrectDate() : PersonException("Некорректная дата!!!") {};

};

class UnknowGender : public PersonException {

public:

    UnknowGender() : PersonException("Неизвестный пол!!! (только M или F)") {};

};

class WrongRelations : public PersonException {

public:

    WrongRelations() : PersonException("Некорректные родственные связи!!!\

    \n1. Если вы устанавливаете мать, то это обязательно должна быть женщина и её  возраст должен быть больше возраста ребёнка\

    \n2. Отец обязательно мужчина и его возраст больше возраста ребёнка\

    \n3. Супруги должны быть разных полов\

    \n4. Дети должны быть младше своих родителей\

    \n5. Так же проверьте, не состоят ли выбранные персоны уже в связях, если

состоят, удалите их и попробуйте заново") {};

};

class ReEntryPerson : public PersonException {

public:

    ReEntryPerson() : PersonException("Такой человек уже есть!!!") {};

};

class NoRelations : public PersonException {

public:

    NoRelations() : PersonException("Родственные отношения для данных персон не

установлены!!!") {};

};

class NotExistPerson : public PersonException {

public:

    NotExistPerson() : PersonException("Такой персоны не существует!!!") {};

};

class UnexpectedEvent : public PersonException {

public:

    UnexpectedEvent() : PersonException("Непредвиденный случай!!!") {};

};

#endif

*Содержание файла main.cpp:*

#include <iostream>

#include "mainMenu.hpp"

using namespace std;

int main() {

    try {

        MainMenu menu;

        menu.callMainMenu();

    }

    catch (PersonException exception) {

        cout << endl << exception.what() << endl;

    }

    system("pause");

    return 0;

}