ГУАП
КАФЕДРА №14

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
ПРЕПОДАВАТЕЛЬ

Должность, уч. степень, звание подпись, дата инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №4

по курсу: ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ

РАБОТУ ВЫПОЛНИЛ
СТУДЕНТ ГР. 1441 М.И. Лубинец

подпись, дата инициалы, фамилия

Санкт-Петербург
2016

1. Формализация задачи

Задача №1:

Составить программу, которая обрабатывает различные виды документов: паспорт, студенческий билет, загранпаспорт. Программа должна узнать у пользователя количество документов, данные о которых он собирается ввести. После этого попросить пользователя для каждого документа (пользователь каждый раз должен указывать тип добавляемого документа) указать необходимые для данного документа данные.

Данные для типов документов:

Паспорт:
- фамилия,
- имя,
- отчество,
- дата рождения (DD.MM.YYYY),
- дата выдачи документа

Студенческий билет:
- фамилия,
- имя,
- название университета,
- номер группы

Загранпаспорт:
- фамилия,
- имя,
- дата рождения,
- дата выдачи документа,
- дата окончания действия

После ввода данных у пользователя должна быть возможность получить информацию о данных введенных документов на экране

При разработке программы использовать ООП подход.

Добавить возможность пользователю производить манипуляции со списком документов из task08win.txt. В частности, должна быть возможность добавить документ нужного типа в список, удалить документ из списка, узнать количество элементов в списке (всего, указанного типа), отобразить документ (заданный, все или заданного типа) на экране, добавить возможность поиска документа по фамилии владельца документа и дате его рождения (учесть, что не каждый тип документа имеет в своих данных дату рождения владельца). Этот инструмент по управлению списком документов должен быть реализован в виде класса.

Добавить возможность сохранения списка документов в файл и чтения списка документов из файла. Для записи и чтения в файл использовать операторы записи в (файловый) поток и чтения из потока.

AbstractDocument.cpp

```
#include "AbstractDocument.hpp"

std::ostream& operator<< (std::ostream& os, const AbstractDocument& obj) {
    obj.write(os);
    return os;
}

std::istream& operator>> (std::istream& is, AbstractDocument& obj) {
    obj.read(is);
    return is;
}

time_t
AbstractDatedDocument::getDate(std::istream& is) {
    tm date {};

    std::string s;
    is >> s;
    std::stringstream ss;
    ss << s;

    char delim = '.';
    for(int i = 0; i < 3; i++) {
        getline(ss, s, delim);
        if(s[0] == '\n') {
            s.erase(s.begin(), s.begin()+1);
        }
        switch(i) {
        case 0: date.tm_mday = std::stoi(s); break;
        case 1: date.tm_mon  = std::stoi(s); delim = '\n'; break;
        case 2: date.tm_year = std::stoi(s); break;
        }
    }
    return timelocal(&date);
}

void
AbstractDatedDocument::printTime(std::ostream& os, time_t time) {
    tm* date = localtime(&time);
    os << std::setw(2) << std::setfill('0') << date->tm_mday << "."
       << std::setw(2) << std::setfill('0') << date->tm_mon << "."
       << date->tm_year
       << " ";
}
```

DocumentFactory.cpp

```
#include "DocumentFactory.hpp"
#include <stdexcept>

std::map<int, CreatorPointer>
DocumentFactory::creatorMap;


const char*
TypeAlteadyRegisteredException::what() const noexcept {
    std::stringstream ss;
    ss << "Type " << type << " is already registered";
    return ss.str().c_str();
}

void
DocumentFactory::push(AbstractDocumentCreator* creator) {
    if(creatorMap.find(creator->type) != creatorMap.end()) {
        throw TypeAlteadyRegisteredException(creator->type);
    }

    creatorMap[creator->type] = CreatorPointer(creator);
}

AbstractDocumentCreator&
DocumentFactory::get(int type) {
    return *creatorMap[type];
}

DocumentPointer
DocumentFactory::create(int type) {
    return creatorMap[type]->operator()();
}

DocumentPointer
DocumentFactory::promt() {
    std::cout << "Availible document types:" << std::endl;
    for(auto& type : creatorMap) {
        auto& dt = *type.second.get();
        std::cout << dt.type << ": " << dt.name << std::endl;
    }

    int type;
    std::cout << "Select type: ";
    std::cin  >> type;

    auto doc = create(type);
    doc->promt();
    return doc;
}
```

DocumentManager.cpp

```
#include "DocumentManager.hpp"
#include <fstream>

DocumentManager::DocumentManager() {}

void
DocumentManager::promt() {
    int n;
    std::cout << "Enter number of your documents: ";
    std::cin  >> n;

    // Create N documents
    for(int i = 0; i < n; i++) {
        std::cout << std::endl << "Document #" << i << std::endl;
        doclist.push_back(DocumentFactory::promt());
    }
}

void
DocumentManager::save(const std::string& filename) const {
    std::ofstream file(filename);

    for(auto& doc : doclist) {
        file << *doc;
    }
}

void
DocumentManager::load(const std::string& filename) {
    std::ifstream file(filename);

    int type;
    while(file >> type) {
        auto doc = DocumentFactory::create(type);
        file >> *doc;
        push(doc);
    }
}

void
DocumentManager::push(const DocumentPointer& doc) {
    doclist.push_back(doc);
}

void
DocumentManager::push(AbstractDocument* docptr) {
    doclist.emplace_back(docptr);
}


void
DocumentManager::del(int type) {
    for(auto it = doclist.begin(); it != doclist.end();) {
        if(it->get()->type == type) {
            it = doclist.erase(it);
        } else {
            ++it;
        }
    }
}

uint
DocumentManager::count() const {
    return static_cast<uint>(doclist.size());
}
```

```
uint
DocumentManager::count(int type) const {
    uint cnt = 0;
    for(auto it = doclist.begin(); it != doclist.end(); ++it) {
        if(it->get()->type == type) {
            cnt++;
        }
    }
    return cnt;
}

void
DocumentManager::show() const {
    for(auto it = doclist.begin(); it != doclist.end(); ++it) {
        std::cout << *it->get() << std::endl;
    }
}

void
DocumentManager::show(int type) const {
    for(auto it = doclist.begin(); it != doclist.end(); ++it) {
        if(it->get()->type == type) {
             std::cout << *it->get() << std::endl;
        }
    }
}

void
DocumentManager::show(const std::string& sname) const {
    for(auto it = doclist.begin(); it != doclist.end(); ++it) {
        if(it->get()->secondName == sname) {
             std::cout << *it->get() << std::endl;
        }
    }
}

void
DocumentManager::show(time_t bd) const {
    for(auto it = doclist.begin(); it != doclist.end(); ++it) {
        if(!it->get()->dated) continue;
        auto datedDocument = dynamic_cast<AbstractDatedDocument*>(it->get());
        if(datedDocument->birthDate == bd) {
             std::cout << *it->get() << std::endl;
        }
    }
}
```

ID.cpp

```
#include "ID.hpp"
#include "DocumentFactory.hpp"
#include <memory>

void
IDCreator::Init() {
    DocumentFactory::push(new IDCreator(DocType_ID, "Passport"));
}

DocumentPointer
IDCreator::operator()() {
    return DocumentPointer(new ID(type));
}

std::ostream&
ID::write(std::ostream& os) const {
    os << type << " "
       << name << " "
       << secondName << " "
       << fatherName << " ";
    ID::printTime(os, birthDate);
    ID::printTime(os, startDate);
    return os;
}

std::istream&
ID::read(std::istream& is) {
    is >> name
       >> secondName
       >> fatherName;
    birthDate = ID::getDate(is);
    startDate = ID::getDate(is);
    return is;
}

void
ID::promt() {
    std::cout << "Name:         ";
    std::cin  >> name;
    std::cout << "Second Name:  ";
    std::cin  >> secondName;
    std::cout << "Father Name:  ";
    std::cin  >> fatherName;
    std::cout << "Birth Date:   ";
    birthDate = ID::getDate(std::cin);
    std::cout << "Start Date:   ";
    startDate = ID::getDate(std::cin);
}
```

InternationalID.cpp

```
#include "InternationalID.hpp"
#include "DocumentFactory.hpp"

void
InternationalIDCreator::Init() {
    DocumentFactory::push(new InternationalIDCreator(DocType_InternetionalID, "International
Passport"));
}

DocumentPointer
InternationalIDCreator::operator()() {
    return DocumentPointer(new InternationalID(type));
}

std::ostream&
InternationalID::write(std::ostream& os) const {
    os << type << " "
       << name << " "
       << secondName << " ";
    InternationalID::printTime(os, birthDate);
    InternationalID::printTime(os, startDate);
    InternationalID::printTime(os, endDate);
    return os;
}

std::istream&
InternationalID::read(std::istream& is) {
    is >> name
       >> secondName;
    birthDate = InternationalID::getDate(is);
    startDate = InternationalID::getDate(is);
    endDate = InternationalID::getDate(is);
    return is;
}

void
InternationalID::promt() {
    std::cout << "Name:         ";
    std::cin  >> name;
    std::cout << "Second Name:  ";
    std::cin  >> secondName;
    std::cout << "Birth Date:   ";
    birthDate = InternationalID::getDate(std::cin);
    std::cout << "Start Date:   ";
    startDate = InternationalID::getDate(std::cin);
    std::cout << "End Date:     ";
    endDate = InternationalID::getDate(std::cin);
}
```

StudentID.cpp

```
#include "StudentID.hpp"
#include "DocumentFactory.hpp"

void
StudentIDCreator::Init() {
    DocumentFactory::push(new StudentIDCreator(DocType_StudentID, "Student ID"));
}

DocumentPointer
StudentIDCreator::operator()() {
    return DocumentPointer(new StudentID(type));
}

std::ostream&
StudentID::write(std::ostream& os) const {
    return os << type << " "
              << name << " "
              << secondName << " "
              << universityName << " "
              << groupNum << " ";
}

std::istream&
StudentID::read (std::istream& is) {
    return is >> name
              >> secondName
              >> universityName
              >> groupNum;
}

void
StudentID::promt() {
    std::cout << "Name:         ";
    std::cin  >> name;
    std::cout << "Second Name:  ";
    std::cin  >> secondName;
    std::cout << "University:   ";
    std::cin  >> universityName;
    std::cout << "Group Number: ";
    std::cin  >> groupNum;
}
```

main.cpp

```
#include <iostream>
#include "DocumentManager.hpp"
#include "ID.hpp"
#include "InternationalID.hpp"
#include "StudentID.hpp"

using namespace std;

int main() {
    IDCreator::Init();
    InternationalIDCreator::Init();
    StudentIDCreator::Init();
    DocumentManager docman;

    docman.promt();

    cout << "\nYou have " << docman.count() << " documents.\n";
    cout << "Your documents: ";
    docman.show();

    for(auto& type : DocumentFactory::creatorMap) {
        auto& doc = *type.second.get();
        cout << "You have " << docman.count(doc.type) << " documents of type " << doc.name << std::endl;
        cout << "Your documents of type " << doc.name << ":" << endl;
        docman.show(doc.type);
        cout << endl;
    }

    cout << "Enter date to search by: ";
    docman.show(AbstractDatedDocument::getDate(cin));

    std::cout << std::endl;

    std::string sname;
    cout << "Enter second name to search by: ";
    cin >> sname;
    docman.show(sname);

    cout << "Documents saved to file \"docs.txt\"\n";
    docman.save("docs.txt");

    for(auto& type : DocumentFactory::creatorMap) {
        auto& doc = *type.second.get();
        docman.del(doc.type);
        cout << "Deleting documents of type " << doc.name << ":" << endl;
        cout << "\nYou have " << docman.count() << " documents.\n";
        cout << endl;
    }

    cout << "Loading documents from file \"docs.txt\"\n";
    docman.load("docs.txt");

    cout << "\nYou have " << docman.count() << " documents.\n";
    cout << "Your documents: ";
    docman.show();

}
```