|  |  |
| --- | --- |
|  | **Технически Университет – София** |
|  |
|  |

***Курсов проект по***

***ООП***

Разработил:

Християн Зарков ФКСТ Проверил:   
Фак. № 501215038 /Иван Станков/

Група: 54

Задание:

**Създайте приложение, което да поддържа информация за издадени учебници. За всеки учебник се пази следната информация: заглавие, автор /творчески колектив/, поредно издание, ISBN номер, дата на излизане от печат и тираж. Учебниците могат да бъдат одобрени от Министерството на образованието, или все още да не са получили този сертификат. Пази се също и дата на одобряване. Работи се с няколко книгоразпространителя. Затова се пази името, адреса и телефона за всеки от книгоразпространителите.**

**Приложението да има възможност за въвеждане на произволен брой различни учебници и книгоразпространители(10 точки).Да има възможност за избор на книгоразпространител и за него да се поръчват различни учебници, като се изчислява общата цена на поръчката(10 точки).**

**Класовете (най-малко 3 класа при реализацията) трябва да капсулира всичките детайли. Използват се private инстанции на променливите за съхраняване на различните детайли. Трябва да има най-малко два конструктора, public getters/setters за private инстанции на променливите (30 точки).**

**Необходимо е да извършвате проверка на входните данни, където е необходимо (10 точки).**

**Да се предефинира операцията <<, която да се използва за извеждане на данните (10 точки). Данните да се четат и съхраняват във файл (20 точки).**

**Класовете да се опишат с UML клас диаграма (10 точки).**

**MAIN**

#include <iostream>

#include <memory>

#include <functional>

#include <regex>

#include "Manual.hpp"

#include "ManualDistributor.hpp"

#include "File.hpp"

class Program

{

private:

FileSystem filesys;

std::vector<std::shared\_ptr<ManualDistributor>> distributors;

std::vector<std::shared\_ptr<Manual>> manuals;

void printMenu()

{

std::cout << std::endl << std::endl;

std::cout << "Menu:" << std::endl;

std::cout << "0. Menu" << std::endl;

std::cout << "1. Add distributor" << std::endl;

std::cout << "2. Add manual" << std::endl;

std::cout << "3. Cetificate manual" << std::endl;

std::cout << "4. Buy manual for distributor" << std::endl;

std::cout << "5. Show manual" << std::endl;

std::cout << "6. Show distributor" << std::endl;

std::cout << "7. Show distributor's cart" << std::endl;

std::cout << "10. Save manuals to file" << std::endl;

std::cout << "11. Save distributors to file" << std::endl;

std::cout << "12. Load manuals from file" << std::endl;

std::cout << "13. Load distributors to file" << std::endl;

std::cout << "100. EXIT" << std::endl << std::endl;

}

template<typename T>

T getData(std::string print, std::function<void(T value)> validator = nullptr)

{

T result;

while(1)

{

std::cout << print << ": ";

std::cin >> result;

bool fail = std::cin.fail();

std::cin.clear();

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

if(fail)

{

std::cout << "cin failed! Bad input." << std::endl;

continue;

}

try

{

if(nullptr != validator)

{

validator(result);

}

break;

}

catch(std::exception& e)

{

std::cout << e.what() << std::endl;

}

}

return result;

}

std::shared\_ptr<ManualDistributor> generateManualDistributor()

{

std::string name, phone, address;

name = this->getData<std::string>("Distributor name", [] (std::string title)

{

if( !std::regex\_match(title, std::regex("[a-zA-Z ]+")) )

{

throw std::invalid\_argument("Incorect name. ");

}

});

phone = this->getData<std::string>("Distributor phone", [] (std::string phone)

{

if( !std::regex\_match(phone, std::regex("^(0|\\+359)[[:digit:]]{9}")) )

{

throw std::invalid\_argument("Incorect phone.");

}

});

address = this->getData<std::string>("Distributor address");

return std::make\_shared<ManualDistributor>(name, phone, address);

}

void addManualDistributor(std::shared\_ptr<ManualDistributor> distributor)

{

this->distributors.push\_back(distributor);

}

std::shared\_ptr<Manual> generateManual()

{

std::string title;

std::string author;

unsigned int edition = 0;

std::string ISBN;

std::tm \*release\_date;

unsigned int price = 0;

title = this->getData<std::string>("Manual title", [] (std::string title)

{

if( !std::regex\_match(title, std::regex("[a-zA-Z ]+")) )

{

throw std::invalid\_argument("Incorect title.");

}

});

author = this->getData<std::string>("Manual author", [] (std::string title)

{

if( !std::regex\_match(title, std::regex("[a-zA-Z ]+")) )

{

throw std::invalid\_argument("Incorect author.");

}

});

edition = this->getData<unsigned int>("Manual edition");

ISBN = this->getData<std::string>("Manual ISBN", [] (std::string ISBN)

{

if( !std::regex\_match(ISBN, std::regex("[[:digit:]]+")) )

{

throw std::invalid\_argument("Incorect ISBN.");

}

});

release\_date = new std::tm;

std::cout << "Manual release date: " << std::endl;

release\_date->tm\_mon = this->getData<int>("Month", [] (int month)

{

if(12 < month || 0 > month)

{

throw std::invalid\_argument("Bad month input");

}

});

release\_date->tm\_year = this->getData<int>("Year", [] (int year)

{

if(0 > year)

{

throw std::invalid\_argument("Bad year input");

}

});

price = this->getData<unsigned int>("Manual's price");

std::shared\_ptr<Manual> ptr = std::make\_shared<Manual>(title, author, edition, ISBN, release\_date, price);

return ptr;

}

void addManual(std::shared\_ptr<Manual> manual)

{

std::cout << "Title:" << manual->getTitle() << std::endl;

this->manuals.push\_back(manual);

}

void printManuals()

{

std::cout << "Manuals: " << std::endl;

for( std::vector<std::shared\_ptr<Manual>>::iterator it = this->manuals.begin(); it != this->manuals.end(); it++ )

{

std::cout << (it - this->manuals.begin()) << ". " << (\*it)->getTitle() << std::endl;

}

}

std::shared\_ptr<Manual> pickManual()

{

if(this->manuals.size() <= 0)

{

throw std::invalid\_argument("No manuals");

}

this->printManuals();

unsigned int number = this->getData<unsigned int>("Number of the manual", [&] (unsigned int number)

{

if(this->manuals.size() < number)

{

throw std::invalid\_argument("Bad manual number!");

}

});

return this->manuals[number];

}

std::tm\* generateCetificateDate()

{

std::tm\* cert\_date = new std::tm;

std::cout << "Certificate Date: " << std::endl;

cert\_date->tm\_mon = this->getData<int>("Month", [] (int month)

{

if(12 < month || 0 > month)

{

throw std::invalid\_argument("Bad month input");

}

});

cert\_date->tm\_year = this->getData<int>("Year", [] (int year)

{

if(0 > year)

{

throw std::invalid\_argument("Bad year input");

}

});

return cert\_date;

}

void printManualDistributors()

{

std::cout << "Manual Distributors: " << std::endl;

for( std::vector<std::shared\_ptr<ManualDistributor>>::iterator it = this->distributors.begin(); it != this->distributors.end(); it++ )

{

std::cout << (it - this->distributors.begin()) << ". " << (\*it)->getName() << std::endl;

}

}

std::shared\_ptr<ManualDistributor> pickManualDistributor()

{

if(this->distributors.size() <= 0)

{

throw std::invalid\_argument("No distributors");

}

this->printManualDistributors();

unsigned int number = this->getData<unsigned int>("Pick number of manual distributor", [&] (unsigned int number)

{

if(this->distributors.size() <= number)

{

throw std::invalid\_argument("Bad manual number!");

}

});

return this->distributors[number];

}

void showDistributor(std::shared\_ptr<ManualDistributor> distributor)

{

std::cout << \*distributor << std::endl;

}

void showManual(std::shared\_ptr<Manual> manual)

{

std::cout << \*manual << std::endl;

}

std::string pickFile()

{

return this->getData<std::string>("File name", [] (std::string file\_name)

{

if( !std::regex\_match(file\_name, std::regex("[a-zA-Z0-9\\-]+\\.data")) )

{

throw std::invalid\_argument("Incorect file name. ");

}

});

}

public:

unsigned int getOption()

{

unsigned int option;

option = this->getData<unsigned int>("Pick Option");

return option;

}

void executeOption(int option)

{

switch(option)

{

case 0:

this->printMenu();

break;

case 1:

this->addManualDistributor(this->generateManualDistributor());

break;

case 2:

this->addManual(this->generateManual());

break;

case 3:

try

{

std::shared\_ptr<Manual> picked\_manual = this->pickManual();

picked\_manual->certificate(this->generateCetificateDate());

}

catch(std::exception& e)

{

std::cout << e.what() << std::endl;

}

break;

case 4:

try

{

std::shared\_ptr<ManualDistributor> picked\_distributor = this->pickManualDistributor();

picked\_distributor->addManual(this->pickManual());

}

catch(std::exception& e)

{

std::cout << e.what() << std::endl;

}

break;

case 5:

try

{

std::shared\_ptr<Manual> picked\_manual = this->pickManual();

this->showManual(picked\_manual);

}

catch(std::exception& e)

{

std::cout << e.what() << std::endl;

}

break;

case 6:

try

{

std::shared\_ptr<ManualDistributor> picked\_distributor = this->pickManualDistributor();

this->showDistributor(picked\_distributor);

}

catch(std::exception& e)

{

std::cout << e.what() << std::endl;

}

break;

case 7:

{

std::shared\_ptr<ManualDistributor> picked\_distributor = this->pickManualDistributor();

std::cout << picked\_distributor->getManualsSum() << std::endl;

}

case 10:

try

{

std::string picked\_file = this->pickFile();

this->filesys.saveToFile<Manual>( picked\_file, this->manuals );

}

catch(std::exception& e)

{

std::cout << e.what() << std::endl;

}

break;

case 11:

try

{

std::string picked\_file = this->pickFile();

this->filesys.saveToFile<ManualDistributor>( picked\_file, this->distributors );

}

catch(std::exception& e)

{

std::cout << e.what() << std::endl;

}

break;

case 12:

try

{

std::string picked\_file = this->pickFile();

auto result\_vector = this->filesys.readFromFile<Manual>( picked\_file );

this->manuals.insert(this->manuals.begin(), result\_vector.begin(), result\_vector.end());

}

catch(std::exception& e)

{

std::cout << e.what() << std::endl;

}

break;

case 13:

try

{

std::string picked\_file = this->pickFile();

auto result\_vector = this->filesys.readFromFile<ManualDistributor>( picked\_file );

this->distributors.insert(this->distributors.begin(), result\_vector.begin(), result\_vector.end());

}

catch(std::exception& e)

{

std::cout << e.what() << std::endl;

}

break;

}

}

};

int main()

{

Program program;

while(1)

{

unsigned int option = 0;

program.executeOption(option);

option = program.getOption();

if(option == 0)

{

continue;

}

if(option == 100)

{

break;

}

program.executeOption(option);

}

}

Manual

#ifndef MANUAL\_HPP

#define MANUAL\_HPP

#include <string>

#include <ctime>

#include <vector>

#include <memory>\

class Manual

{

private:

std::string title;

std::string author;

unsigned int edition;

std::string ISBN;

std::tm \*release\_date;

std::tm \*cert;

unsigned int price;

//unsigned int edition;

public:

Manual( Manual& copy )

{

this->title = copy.title;

this->author = copy.author;

this->edition = copy.edition;

this->ISBN = copy.ISBN;

this->release\_date = copy.release\_date;

this->cert = cert;

}

Manual( std::string title, std::string author, unsigned int edition, std::string ISBN, std::tm \*release\_date, unsigned int price )

:title(title), author(author), edition(edition), ISBN(ISBN), release\_date(release\_date),cert(NULL), price(price)

{

}

Manual()

:title(""), author(""), edition(0), ISBN(""), release\_date(NULL), cert(NULL), price(0)

{

}

~Manual()

{

//delete time

}

std::string getTitle()

{

return this->title;

}

void setTitle( std::string title )

{

this->title = title;

}

std::string getAuthor()

{

return this->author;

}

void setAuthor( std::string author )

{

this->author = author;

}

unsigned int getEdition()

{

return this->edition;

}

void setEdition( unsigned int edition )

{

this->edition = edition;

}

std::string getISBN()

{

return this->ISBN;

}

void setISBN( std::string ISBN )

{

this->ISBN = ISBN;

}

std::tm \*getReleaseDate()

{

return this->release\_date;

}

void setReleaseDate( std::tm\* release\_date )

{

this->release\_date = release\_date;

}

bool isCertificared()

{

if(NULL == this->cert)

{

return false;

}

return true;

}

void certificate( std::tm \*date = NULL )

{

this->cert = date;

}

unsigned int getPrice()

{

return this->price;

}

void setPrice( unsigned int price )

{

this->price = price;

}

/\*unsigned int getEdition()

{

return this->edition;

}\*/

bool operator==(const Manual& manual) const

{

return this->title == manual.title;

}

bool operator!=(const Manual& manual) const

{

return !((\*this) == manual);

}

bool operator==(const std::string& title) const

{

return this->title == title;

}

bool operator!=(const std::string& title) const

{

return !((\*this) == title);

}

friend std::ostream& operator<<(std::ostream& os, Manual& manual);

friend std::istream& operator>>(std::istream& is, Manual& manual);

};

std::tm\* stringToTime(std::string time)

{

std::tm\* result = new std::tm;

size\_t delimiter = time.find("/");

result->tm\_mon = std::stoi(time.substr(0, delimiter));

result->tm\_year = std::stoi(time.substr(delimiter+1));

return result;

}

std::ostream& operator<<(std::ostream& os, Manual& manual)

{

//<< "Manual" << ","

os << manual.title << ","

<< manual.author << ","

<< manual.edition << ","

<< manual.ISBN << ","

<< manual.release\_date->tm\_mon << "/" << manual.release\_date->tm\_year << ","

<< (manual.isCertificared() ? (manual.cert->tm\_mon + "/" + manual.cert->tm\_year):"NO") << ","

<< manual.price

<< std::endl;

return os;

}

std::istream& operator>>(std::istream& is, Manual& manual)

{

std::string substr;

is >> substr;

if(is.fail())

{

return is;

}

std::stringstream ss( substr );

getline( ss, substr, ',' );

manual.title = substr;

getline( ss, substr, ',' );

manual.author = substr;

getline( ss, substr, ',' );

manual.edition = std::stoi(substr);

getline( ss, substr, ',' );

manual.ISBN = substr;

getline( ss, substr, ',' );

manual.release\_date = stringToTime(substr);

getline( ss, substr, ',' );

if("NO" != substr)

{

manual.cert = stringToTime(substr);

}

getline( ss, substr );

manual.price = std::stoi(substr);

return is;

}

#endif

ManualDistributor

#ifndef MANUAL\_DISTRIBUTOR\_HPP

#define MANUAL\_DISTRIBUTOR\_HPP

#include <string>

#include <algorithm>

#include "Manual.hpp"

class ManualDistributor

{

private:

std::string name;

std::string phone;

std::string address;

std::vector<std::shared\_ptr<Manual>> manuals;

public:

ManualDistributor()

:name(""),phone(""),address("")

{

}

ManualDistributor(std::string name, std::string phone, std::string address)

:name(name), phone(phone), address(address)

{

}

void setName(std::string name)

{

this->name = name;

}

void setPhone(std::string phone)

{

this->phone = phone;

}

void setAddress(std::string address)

{

this->address = address;

}

std::string getName()

{

return this->name;

}

std::string getPhone()

{

return this->phone;

}

std::string getAddress()

{

return this->address;

}

bool operator==(const ManualDistributor& manualdist) const

{

return this->name == manualdist.name;

}

bool operator!=(const ManualDistributor& manualdist) const

{

return !((\*this) == manualdist);

}

bool operator==(const std::string& name) const

{

return this->name == name;

}

bool operator!=(const std::string& name) const

{

return !((\*this) == name);

}

friend std::ostream& operator<<(std::ostream& os, ManualDistributor& distributor);

friend std::istream& operator>>(std::istream& is, ManualDistributor& distributor);

void addManual(std::shared\_ptr<Manual> manual)

{

if( this->manuals.end() != std::find\_if(this->manuals.begin(), this->manuals.end(),

[&] (std::shared\_ptr<Manual> const& p)

{

return \*p == \*manual;

})

)

{

throw std::invalid\_argument("This manual is already added for this distributor");

}

this->manuals.push\_back(manual);

}

unsigned int getManualsSum()

{

unsigned int sum = 0;

for( std::vector<std::shared\_ptr<Manual>>::iterator it = this->manuals.begin(); it != this->manuals.end(); it++ )

{

sum += (\*it)->getPrice();

}

return sum;

}

};

std::ostream& operator<<(std::ostream& os, ManualDistributor& distributor)

{

os << distributor.name << ","

<< distributor.phone << ","

<< distributor.address

<< std::endl;

return os;

}

std::istream& operator>>(std::istream& is, ManualDistributor& distributor)

{

std::string substr;

is >> substr;

if(is.fail())

{

return is;

}

std::stringstream ss( substr );

getline( ss, substr, ',' );

distributor.name = substr;

getline( ss, substr, ',' );

distributor.phone = substr;

getline( ss, substr );

distributor.address = substr;

return is;

}

#endif

**FileSystem**

#include <iostream>

#include <fstream>

#include <vector>

#include <memory>

class FileSystem

{

private:

public:

template<typename InputType, typename SharedType = std::shared\_ptr<InputType>>

void saveToFile(std::string path, std::vector<SharedType> input\_vector)

{

std::ofstream FILE(path, std::ios::out | std::ofstream::binary);

if(!FILE.is\_open())

{

throw std::runtime\_error("Failed to open " + path);

}

for( typename std::vector<SharedType>::iterator it = input\_vector.begin(); it != input\_vector.end(); it++ )

{

FILE << \*(\*it);

}

}

template<typename InputType, typename SharedType = std::shared\_ptr<InputType>>

std::vector<SharedType> readFromFile(std::string path)

{

std::vector<SharedType> result;

std::ifstream FILE(path, std::ios::in | std::ifstream::binary);

if(!FILE.is\_open())

{

throw std::runtime\_error("Failed to open " + path);

}

while(1)

{

InputType man;

FILE >> man;

if(FILE.eof())

{

break;

}

if(FILE.fail())

{

continue;

}

result.push\_back(std::make\_shared<InputType>(man));

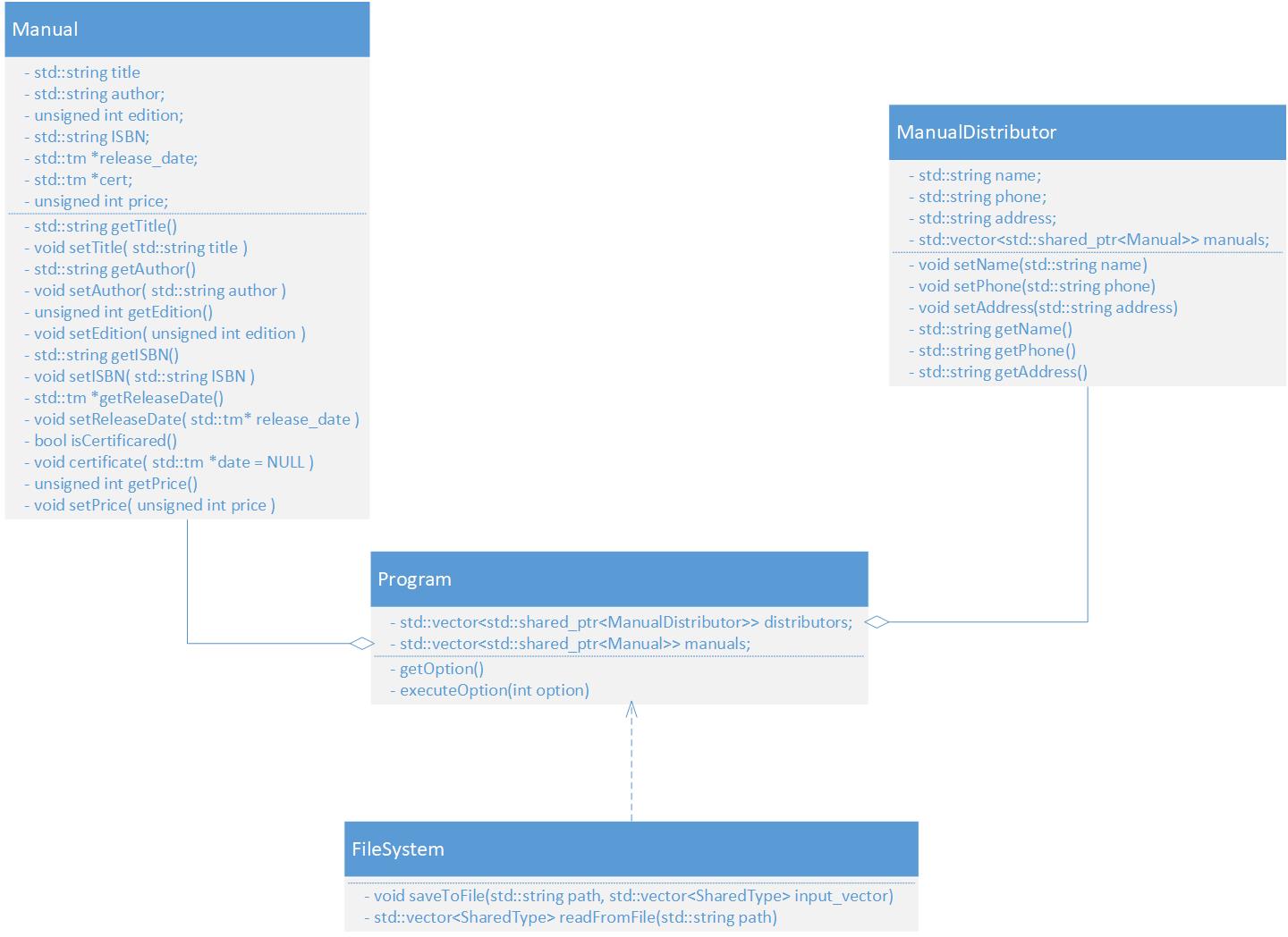
}

return result;

}

};

**UML class diagram**

****