МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное учреждения высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт компьютерных технологий и информационной безопасности

Кафедра математического обеспечения и применения ЭВМ

**ЛАБОРАТОРНАЯ РАБОТА № 4**

по дисциплине

**«Объектно-ориентированное программирование»**

на тему:

**«Контейнеры STL»**

*Вариант № 12*

Выполнил:

Студент группы

КТбо2-8

Данильченко Т. В.

Проверил:

доцент кафедры

МОП ЭВМ

Тарасов С. А.

Оценка

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

Таганрог 2020

# **1 Цель работы**

Ознакомление с механизмом выбора типа создаваемых объектов во время выполнения программы.

# **2 Задание, вариант № 12**

# Создать класс «Библиотека (Library)» с полями: ФИО автора книги*,* название, год издания, количество экземпляров данной книги в библиотеке. Вывести сведения о всех книгах заданного автора, начиная с заданного года издания,

# **3 Ход работы**

**3.1 Спецификации классов**

Диаграмма используемых классов приведена на Рисунке 1.

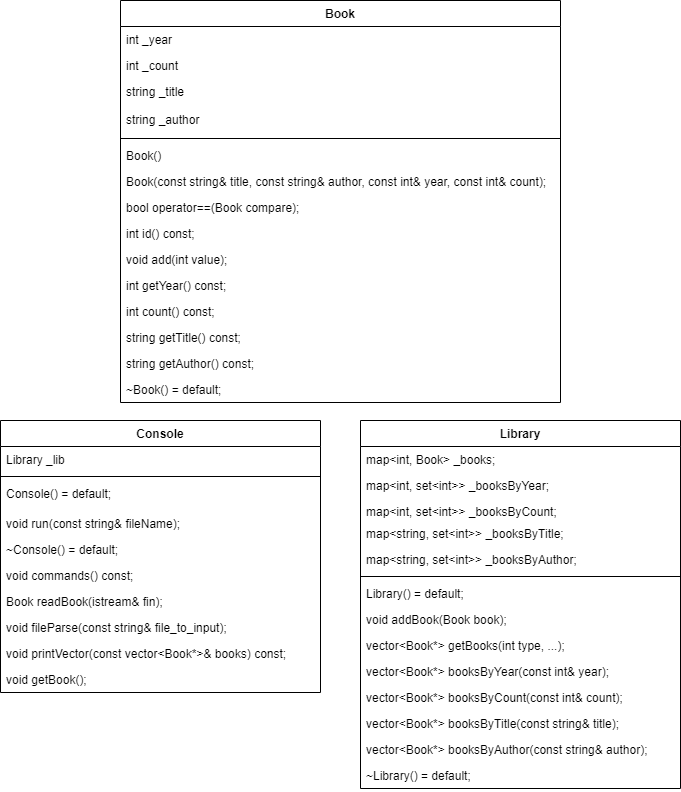


Рисунок 1 — Диаграмма классов

В классе Library определен map книг, хранящихся в библиотеке. Так же, помимо этого map’a есть еще 4 map’a, в которых книги хранятся по определенным параметрам (по году выпуска, по количеству, по названию, по ФИО автора).

В классе Console выполняется взаимодействие с пользователем.

**3 Листинг**

// Book.h

#pragma once

#include <string>

#include <algorithm>

#include <iterator>

using std::sort;

using std::string;

using std::set\_intersection;

class Book {

public:

Book() = default;

Book(const string& title, const string& author, const int& year, const int& count);

bool operator==(Book compare);

int id() const;

void add(int value);

int getYear() const;

int count() const;

string getTitle() const;

string getAuthor() const;

~Book() = default;

private:

int \_year;

int \_count;

string \_title;

string \_author;

int \_id;

static int \_idCounter;

};

// Library.h

#pragma once

#include "Book.h"

#include <map>

#include <set>

#include <vector>

using std::map;

using std::set;

using std::vector;

class Library{

public:

Library() = default;

void addBook(Book book);

vector<Book\*> getBooks(int type, ...);

vector<Book\*> booksByYear(const int& year);

vector<Book\*> booksByCount(const int& count);

vector<Book\*> booksByTitle(const string& title);

vector<Book\*> booksByAuthor(const string& author);

~Library() = default;

private:

map<int, Book> \_books;

map<int, set<int>> \_booksByYear;

map<int, set<int>> \_booksByCount;

map<string, set<int>> \_booksByTitle;

map<string, set<int>> \_booksByAuthor;

};

// Console.h

#pragma once

#include "Library.h"

#include <vector>

#include <string>

#include <fstream>

#include <iostream>

using std::cin;

using std::cout;

using std::string;

using std::istream;

using std::ifstream;

class Console {

public:

Console() = default;

void run(const string& fileName);

~Console() = default;

private:

Library \_lib;

void commands() const;

Book readBook(istream& fin);

void fileParse(const string& file\_to\_input);

void printVector(const vector<Book\*>& books) const;

void getBook();

};

// Book.cpp

#include "Book.h"

int Book::\_idCounter = 1;

Book::Book(const string& title, const string& author, const int& year, const int& count) :

\_title(title), \_author(author), \_year(year), \_count(count), \_id(\_idCounter++)

{}

bool Book::operator==(Book compare) {

return (\_title == compare.\_title

&& \_author == compare.\_author

&& \_year == compare.\_year);

}

int Book::id() const {

return \_id;

}

void Book::add(int value) {

\_count += value;

return;

}

int Book::getYear() const {

return \_year;

}

int Book::count() const {

return \_count;

}

string Book::getTitle() const {

return \_title;

}

string Book::getAuthor() const {

return \_author;

}

// Library.cpp

#include "Library.h"

void Library::addBook(Book book) {

\_books.insert({ book.id(), book });

auto itYear = \_booksByYear.find(book.getYear());

auto itCount = \_booksByCount.find(book.count());

auto itTitle = \_booksByTitle.find(book.getTitle());

auto itAuthor = \_booksByAuthor.find(book.getAuthor());

if (itYear != \_booksByYear.end()) {

itYear->second.insert(book.id());

}

else {

\_booksByYear[book.getYear()] = set<int>();

\_booksByYear[book.getYear()].insert(book.id());

}

if (itCount != \_booksByCount.end()) {

itCount->second.insert(book.id());

}

else {

\_booksByCount[book.count()] = set<int>();

\_booksByCount[book.count()].insert(book.id());

}

if (itTitle != \_booksByTitle.end()) {

itTitle->second.insert(book.id());

}

else {

\_booksByTitle[book.getTitle()] = set<int>();

\_booksByTitle[book.getTitle()].insert(book.id());

}

if (itAuthor != \_booksByAuthor.end()) {

itAuthor->second.insert(book.id());

}

else {

\_booksByAuthor[book.getAuthor()] = set<int>();

\_booksByAuthor[book.getAuthor()].insert(book.id());

}

return;

}

vector<Book\*> Library::getBooks(int type, ...) {

vector<Book\*> books;

va\_list p;

\_\_crt\_va\_start(p, type);

switch (type) {

case 1: {

string title = \_\_crt\_va\_arg(p, string);

books = booksByTitle(title);

break;

}

case 2: {

string author = \_\_crt\_va\_arg(p, string);

books = booksByAuthor(author);

break;

}

case 3: {

int year = \_\_crt\_va\_arg(p, int);

books = booksByYear(year);

break;

}

case 4: {

int count = \_\_crt\_va\_arg(p, int);

books = booksByCount(count);

break;

}

default:

break;

}

\_\_crt\_va\_end(p);

return books;

}

vector<Book\*> Library::booksByYear(const int& year) {

vector<Book\*> books;

auto it = \_booksByYear.find(year);

if (it == \_booksByYear.end()) {

return books;

}

for (set<int>::iterator itSet = it->second.begin(); itSet != it->second.end(); ++itSet) {

books.push\_back(&\_books[\*itSet]);

}

return books;

}

vector<Book\*> Library::booksByCount(const int& count) {

vector<Book\*> books;

auto it = \_booksByCount.find(count);

if (it == \_booksByCount.end()) {

return books;

}

for (set<int>::iterator itSet = it->second.begin(); itSet != it->second.end(); ++itSet) {

books.push\_back(&\_books[\*itSet]);

}

return books;

}

vector<Book\*> Library::booksByTitle(const string& title) {

vector<Book\*> books;

auto it = \_booksByTitle.find(title);

if (it == \_booksByTitle.end()) {

return books;

}

for (set<int>::iterator itSet = it->second.begin(); itSet != it->second.end(); ++itSet) {

books.push\_back(&\_books[\*itSet]);

}

return books;

}

vector<Book\*> Library::booksByAuthor(const string& author) {

vector<Book\*> books;

auto it = \_booksByAuthor.find(author);

if (it == \_booksByAuthor.end()) {

return books;

}

for (set<int>::iterator itSet = it->second.begin(); itSet != it->second.end(); ++itSet) {

books.push\_back(&\_books[\*itSet]);

}

return books;

}

// Console.cpp

#include "Library.h"

void Library::addBook(Book book) {

\_books.insert({ book.id(), book });

auto itYear = \_booksByYear.find(book.getYear());

auto itCount = \_booksByCount.find(book.count());

auto itTitle = \_booksByTitle.find(book.getTitle());

auto itAuthor = \_booksByAuthor.find(book.getAuthor());

if (itYear != \_booksByYear.end()) {

itYear->second.insert(book.id());

}

else {

\_booksByYear[book.getYear()] = set<int>();

\_booksByYear[book.getYear()].insert(book.id());

}

if (itCount != \_booksByCount.end()) {

itCount->second.insert(book.id());

}

else {

\_booksByCount[book.count()] = set<int>();

\_booksByCount[book.count()].insert(book.id());

}

if (itTitle != \_booksByTitle.end()) {

itTitle->second.insert(book.id());

}

else {

\_booksByTitle[book.getTitle()] = set<int>();

\_booksByTitle[book.getTitle()].insert(book.id());

}

if (itAuthor != \_booksByAuthor.end()) {

itAuthor->second.insert(book.id());

}

else {

\_booksByAuthor[book.getAuthor()] = set<int>();

\_booksByAuthor[book.getAuthor()].insert(book.id());

}

return;

}

vector<Book\*> Library::getBooks(int type, ...) {

vector<Book\*> books;

va\_list p;

\_\_crt\_va\_start(p, type);

switch (type) {

case 1: {

string title = \_\_crt\_va\_arg(p, string);

books = booksByTitle(title);

break;

}

case 2: {

string author = \_\_crt\_va\_arg(p, string);

books = booksByAuthor(author);

break;

}

case 3: {

int year = \_\_crt\_va\_arg(p, int);

books = booksByYear(year);

break;

}

case 4: {

int count = \_\_crt\_va\_arg(p, int);

books = booksByCount(count);

break;

}

default:

break;

}

\_\_crt\_va\_end(p);

return books;

}

vector<Book\*> Library::booksByYear(const int& year) {

vector<Book\*> books;

auto it = \_booksByYear.find(year);

if (it == \_booksByYear.end()) {

return books;

}

for (set<int>::iterator itSet = it->second.begin(); itSet != it->second.end(); ++itSet) {

books.push\_back(&\_books[\*itSet]);

}

return books;

}

vector<Book\*> Library::booksByCount(const int& count) {

vector<Book\*> books;

auto it = \_booksByCount.find(count);

if (it == \_booksByCount.end()) {

return books;

}

for (set<int>::iterator itSet = it->second.begin(); itSet != it->second.end(); ++itSet) {

books.push\_back(&\_books[\*itSet]);

}

return books;

}

vector<Book\*> Library::booksByTitle(const string& title) {

vector<Book\*> books;

auto it = \_booksByTitle.find(title);

if (it == \_booksByTitle.end()) {

return books;

}

for (set<int>::iterator itSet = it->second.begin(); itSet != it->second.end(); ++itSet) {

books.push\_back(&\_books[\*itSet]);

}

return books;

}

vector<Book\*> Library::booksByAuthor(const string& author) {

vector<Book\*> books;

auto it = \_booksByAuthor.find(author);

if (it == \_booksByAuthor.end()) {

return books;

}

for (set<int>::iterator itSet = it->second.begin(); itSet != it->second.end(); ++itSet) {

books.push\_back(&\_books[\*itSet]);

}

return books;

}